The application of Probabilistic Graphical Models to Raptor codes over Binary Input Memoryless Symmetric Channel models

by

Rian Singels

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electric and Electronic Engineering in the Faculty of Engineering at Stellenbosch University

Supervisors:

Prof. Johan A. du Preez

Dr. Riaan Wolhuter

March 2016

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: .....

Abstract

The application of Probabilistic Graphical Models to Raptor codes over Binary Input Memoryless Symmetric Channel models

R. Singels

Department of Electric and Electronic Engineering,
University of Stellenbosch,
Private Bag X1, Matieland 7602, South Africa.

Thesis: MScEng (Elec & Elect)

January 2016

Raptor codes are Forward Error Correction (FEC) codes that fall under the class of Fountain codes. This class of code can reach data-transmission rates close to the capacity of the Binary Erasure Channels (BECs). It has consequently been researched and refined for deterministic decoding over these channels. Raptor codes are ideal for communication over the internet as the internet is a realisation of the BEC. This work investigates the use of Raptor codes for probabilistic decoding, assessing their performance over the Binary Symmetric Channel (BSC) and the Binary Additive White Gaussian Noise Channel (BAWGNC).

Extensive consideration is given to the Belief Propagation (BP) algorithm and Probabilistic Graphical Models (PGMs) - tools of inference that are essential to the decoding of FECs codes. Focus is given to the application of the Factor Graph (FG), the Cluster graph (CG), and the Junction Tree (JT). Furthermore, attention is given to how the BP-update rules may be transformed in order to avoid computation over large distributions. The way in which two graph-altering algorithms may improve the decoding success rate, i.e., the Tree-structure Expectation Propagation (TEP) and the Inactivation

Decoding (ID) algorithms, is also shown. These algorithms are simulated and the results analysed.

Uittreksel

Die toepassing van waarskeinlikheidsgrafiese modele op Raptor-kodes oor binêre inset geheuelose simmetriese kanale.

("The application of Probabilistic Graphical Models to Raptor codes over the Binary Input Memoryless Symmetric Channel models")

R. Singels

Departement Elektriese en Elektroniese Ingenieurswese,
Universiteit van Stellenbosch,
Privaatsak X1, Matieland 7602, Suid Afrika.

Tesis: MScIng (Elek. & Elekt.)

Januarie 2016

Raptor-kodes is 'n tipe voorwaarde-foutkorreksiekode wat as 'n Fontein-kode geklassifiseer word. Hierdie klas van kodes kan datatransmissie-tempos na aan die kapasiteit van die binêre afskawing-kanale bereik. Dit is gevolglik nagevors en verfyn vir bepalingsdekodering oor hierdie klas van kanale. Raptor-kodes is ideaal vir kommunikasie oor die internet aangesien die internet 'n realisasie van die binêre afskawing-kanaal verteenwoordig. Hierdie werk ondersoek die gebruik van Raptor-kodes vir waarskynlikheidsdekodering en evalueer sy prestasie oor die binêre simmetriese kanaal en die binêre-toevoeging wit Gaussiese ruis-kanaal.

Uitgebreide oorweging is gebied aan die oortuigingsvoortplanting-algoritme en waarskeinlikheidsgrafiese modelle; instrumente van afleiding wat noodsaaklik vir die dekodering van voorwaarde-foutkorreksie-kodes is. Fokus word gebied aan die toepassing van die faktorgrafiek, die bundelgrafiek, en die aansluitingsboom. Verder word dit behandel hoe die oortuigingsvoortplanting-aanpassingsreëls omskep kan word ten einde berekening oor groot uitkerings

te vermy. Dit word ook behandel hoe twee grafiekverandering-algoritmes die dekodering-suksessyfer kan verbeter, dit wil sê, die boom-gewysde verwagtings-voortplanting- en deaktiveringsdekodering-algoritmes. Hierdie algoritmes is gesimuleer en die resultate is ontleed.

Acknowledgements

I would like to express my sincere gratitude to the following people:

- God for guiding and inspiring me.
- Prof. Johan du Preez, my study leader, for his wisdom, guidance and passion for my work.
- Dr. Riaan Wolhuter for his help.
- Rupert Neethling for his efforts and support regarding the language of this thesis.
- Jaco du Toit for sharing his insights of Raptor codes with me.
- All my friends and colleagues from the DSP lab.
- My parents for constantly supporting and encouragement me during the course of this thesis.

Contents

List of Figures

List of Tables

List of Abbreviations

3GPP	3rd Generation Partnership
a.k.a.	also known as
BAWGNC	Binary Additive White Gaussian Noise Channel
BEC	Binary Erasure Channel
BER	Bit Error Rate
BIMSC	Binary Input Memoryless Symmetric Channel
BSC	Binary Symmetric Channel
BN	Bayes Network
BP	Belief Propagation
BRV	Binary Random Variable
CG	Cluster graph
e.g.	for the sake of example (Latin: *exempli gratia*)
DVB	Digital Video Broadcasting
DRV	Discrete Random Variable
ESI	Encoded Symbol Identifier
etc.	and the rest (Latin: *et cetera*)
et al.	and others
FEC	Forward Error Correction
FG	Factor Graph
GE	Gaussian Elimination
HDPC	High-Density Parity-Check
i.a.	among other things (Latin: *inter alia*)
IETF	Internet Engineering Task Force
ID	Inactivation Decoding
i.e.	that is (Latin: *id est*)
JT	Junction Tree
LDPC	Low-Density Parity-Check
LLR	Log-Likelihood Ratio

LT	Luby Transform
MBMS	Multimedia Broadcast/Multicast Service
MRF	Markov Random Field
NP	Non-deterministic Polynomial-time hard
PGM	Probabilistic Graphical Model
PD	Probability Distribution
PDF	Probability Density Function
RIP	Running Intersection Property
R10	Raptor 10
RQ	RaptorQ
SNR	Signal-to-Noise Ratio
TEP	Tree-structure Expectation Propagation
VE	Variable Elimination
XOR	eXclusive-OR

List of Symbols

Variables

ϵ	Channel erasure probability	$[0-1]$
ρ	Channel bit flip probability	$[0-1]$
σ_n	The standard deviation of Gaussian noise	$[0-\infty]$
${\sigma_n}^2$	The variance of Gaussian noise	$[0-\infty]$
A	The amplitude of the BAWGNC's signal	$[\in(-\infty,\infty)]$
δ	Failure probability of the LT-process	$[0<\delta<<1]$
ω_c	LDPC column weight	$[\geq 3]$
ω_r	LDPC row weight	$[>0]$
$\mathcal{B}$	The number of channel outputs	$[1-\infty]$
$\mathcal{H}$	The number of high density parity symbols	$[1-\infty]$
$\mathcal{I}_{max}$	The maximum allowed number of BP iterations	$[1-\infty]$
$\mathcal{K}$	The number of source symbols	$[1-\infty]$
$\mathcal{L}$	The number of low density parity symbols	$[1-\infty]$
$\mathcal{N}$	The number of input symbols	$[\mathcal{K}+\mathcal{L}]$
$\mathcal{M}$	The number of output symbols	$[1-\infty]$
$\mathcal{D}$	The degree of a node	$[1-\infty]$
$\mathcal{D}_\mathcal{M}$	The degree of a factor node	$[1-\mathcal{N}]$
$\mathcal{D}_\mathcal{N}$	The degree of a variable node	$[1-\mathcal{W}]$
$\mathcal{D}_i$	The input symbol degree of a factor	$[1-\mathcal{M}]$
$\mathcal{D}_o$	The degree of an output symbol	$[1-\mathcal{N}]$
$\mathcal{Q}$	The chosen convergence threshold of a graph	$[0-1]$
$\mathcal{R}$	Rate of data transmission	$[\mathcal{K}/\mathcal{N}]$
$\mathcal{S}$	The number of source bits	$[1-\infty]$

$\mathcal{V}$	The observer value of an output symbol . . . . . $[0-1]$
$\mathcal{W}$	The number of factors or clusters . . . . . $[1-\infty]$

Arrays an matrices

$\mathbf{z}$	Source messages/source symbol set (Original data)
$\mathbf{t}$	Transmitted messages/output symbol set (Encoded data)
$\mathbf{y}$	Input symbol set (Encoded data)
$\mathbf{n}$	Channel noise
$\mathbf{r}$	Received messages/received symbol set (Encoded data and noise)
$\mathbf{s}$	Syndrome vector of a block code
$\mathbf{G}$	The generator matrix of a code
$\mathbf{H}$	The parity check matrix of a code

Symbols

ε	A very small value
z	Single binary source symbol (The original data)
p	Single binary parity symbol
y	Single binary input symbol
t	Single binary output symbol
r	Single observed symbol (Gained at receiver)
x	A collective symbol representing ether a binary source (z), parity (p) or output (r) symbol
s	Single syndrome token of a code
$\lvert X\rvert$	The cardinality of ensemble X
$\mathcal{C}$	A general noisy channel

Functions

$O(X)$	The order of complexity X
S	$c\ln(\frac{\mathcal{K}}{\delta})\sqrt{\mathcal{K}}$
$N(\mu, {\sigma_n}^2)$	Normal distribution: $\frac{1}{\sqrt{2\pi{\sigma_n}^2}}e^{-\frac{(r-\mu)^2}{2{\sigma_n}^2}}$

$P(x)$	The probability distribution of variable x
$P(\hat{x})$	The probability of the outcome $\hat{x}$ of variable x
$h(\hat{x})$	Shannon information content of the outcome $\hat{x}$
$H(X)$	The marginal entropy of ensemble X
$H_2(X)$	The binary entropy of ensemble X
$H(X,Y)$	The joint entropy of ensembles X and Y
$H(X\|Y)$	The conditional entropy of ensemble X given Y
$I(X;Y)$	The mutual information between ensembles X and Y
$\text{Cap}(\mathcal{C})$	The capacity of channel $\mathcal{C}$
$\phi(\mathbf{x})$	A cluster pertaining to the variable set $\mathbf{x}$
$f(\mathbf{x})$	A factor pertaining to the variable set $\mathbf{x}$
$\psi(\mathbf{x})$	A separator pertaining to the variable set $\mathbf{x}$
$\text{Scp}(\phi)$	The scope of cluster ϕ (may also be of f or ψ)
$L(x)$	The LLR of variable x
$\mu_{x_i \to x_j}(x_j)$	A message from node x_i to node x_j
$\lambda_{x_i \to x_j}(x_j)$	Equivalent to $L\Big(\mu_{x_m \to x_n}(x_n)\Big)$

Sets

$\varnothing$	An empty set
$\mathcal{A}_X$	Possible outcomes of Discrete Random Variable (DRV) x
$\mathcal{P}_X$	Probabilities of respective possible values of outcome $\hat{x}$
$\upsilon(x_i)$	The dependencies of x_i, e.g., $P\Big(x_i\|\upsilon(x_i)\Big)$

Operations

$f \stackrel{\text{def}}{=} g$	Define function f to be equivalent to function g
$x \propto y$	x is proportional to y
$\mathbf{x} \subset \mathbf{y}$	$\mathbf{x}$ is a subset of $\mathbf{y}$
$\mathbf{x} \subseteq \mathbf{y}$	$\mathbf{x}$ is a subset of, or equal set to $\mathbf{y}$
$\mathbf{x} \cap \mathbf{y}$	The intersect of the sets $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{x} \cup \mathbf{y}$	The unison of the sets $\mathbf{x}$ and $\mathbf{y}$
$\mathbf{x} \| \mathbf{y}$	The concatenation of the sets $\mathbf{x}$ and $\mathbf{y}$

$x \oplus y$	The modulo 2 addition (XOR) of x and y
$L(x) \boxplus L(y)$	Equivalent to $L(x \oplus y)$
$\mathbf{x} \setminus y$	The set $\mathbf{x}$ exclusion symbol y, where $y \in \mathbf{x}$
$\bigoplus_{i=0}^{N} x_i$	XOR summation, i.e., $x_0 \oplus x_1 \oplus \cdots \oplus x_N$
$\boxplus_{i=0}^{N} L(x_i)$	Box summation, i.e., $L(x_0) \boxplus L(x_1) \boxplus \cdots \boxplus L(x_N)$

Super- and Subscripts

$\hat{x}$	The outcome or estimation of a variable x
i	The i^{th} instance in an array/vector
$x_i \in \mathbf{x} \setminus x_n$	Iterate over every element x_i of set $\mathbf{x}$ excluding x_n

Chapter 1

Introduction

1.1 Background

Fast data transmission is in ever growing demand in the 21st century. The pursuit of data transmission rates close to the theoretical limit of a given medium, i.e., its channel capacity, has met with only limited success. Until recently, it was considered an infeasible endeavour [4]. However, in May 1993, a revolution occurred in coding theory with the release of a paper by C. Berrou [5] that, almost by accident, combined sparse-graph codes with low-complexity iterative decoding, thereby changing the way we approach error correcting codes. Today we have many codes that perform close to the channel capacity and that may be implemented with *low-complexity* decoding algorithms. This includes codes such as the turbo code, the convolution code, and variations of block codes such as the Low-Density Parity-Check (LDPC) codes [6, 7]. The challenge has been that these codes have design difficulties due to their fixed code rates, i.e., a fixed number of bits is required to be transmitted for any given number of original data bits. Furthermore, in some cases all packages must be received and must be received in order. These restrictions render them unsuitable for broadcasting, that is, single transmitter to multiple receivers [8].

A class of codes known as fountain codes has been designed specifically not to suffer from these restrictions. The "digital fountain approach" was first introduced in [9] for communication over a channel, not with data corruption, but with data loss known as erasures. What makes fountain codes appropriate for broadcasting is that they are naturally *rateless*, i.e., for a given finite set of data, a fountain code can theoretically generate an infinite sequence of

encoded data for transmission. This is done while retaining low complexity for both the encoding and decoding and therefore remaining viable codes for time constrained applications.

The first practical realisation of a fountain code was the Luby Transform (LT)-code as introduced by M. Luby in [10] for Binary Erasure Channel (BEC), which is a channel with binary inputs and outputs with some data loss. Unfortunately for reliable decoding, the length of the LT-code increases greatly with small increases of the code length.

An extension of the LT-code called the Raptor (*rap*id-*tor*nado) code improves on it by first encoding the original data with a sparse-graph code before encoding the resulting data set again with the LT-code. This solves the problem of super-linear growth and yields linear time encoders and decoders [11]. The Raptor code was invented in late 2000 and patented in 2001 [12]. Since then it has been adopted into a number of different standards. These include 3rd Generation Partnership (3GPP) Multimedia Broadcast/Multicast Service (MBMS) [13], Internet Engineering Task Force (IETF) [14] and many others [8].

1.2 Motivation for this Work

All but a few of the existing works on the Raptor code are focused on its development for the BEC. This can be attributed to existing architectures, such as the internet, that behave much like the BEC models and that other error correcting codes are less suited to than fountain codes for use over a BEC. Another contributing factor is that the complexity of decoding over a BEC is also relatively low, thereby allowing for real-time decoding.

Raptor codes are, however, not restricted to the BEC model and, in combination with the most current Probabilistic Graphical Model (PGM) techniques, have the potential to be the versatile tool that may bring us closer to the Shannon limit for general broadcasting applications. It is for this reason that we wish to investigate the application of Raptor codes using these PGMs.

1.3 Project Objectives

Considering that most of the work on the Raptor code is focused on its development for the BEC model, it was our objective to investigate the application of the Raptor code on other channel models. In addition we aimed to apply the latest PGM-architectures and decoding techniques to the Raptor code and compare their differences in performance and complexity for this application.

Thus the objectives of the work documented herein were defined as follows:

- To analyse the design and structure of Raptor codes.
- To apply Raptor codes to channel models other than the BEC.
- To investigate the suitability of a variety of PGMs for the decoding of Raptor codes.

1.4 Project Outcomes

The contributions made in this project are as follows:

Theory mastered

- Definitions of the BEC, Binary Symmetric Channel (BSC), and Binary Additive White Gaussian Noise Channel (BAWGNC) channel models were given.
- The theoretical limits of the rate at which information can be reliably transferred over these channels were presented using the information theory of Entropy.
- The design of Raptor codes and their structure was described.
- An investigation of the construction and use of a variety of PGMs was done - specifically on the Bayes Network (BN), Markov Random Field (MRF), Factor Graph (FG), Cluster graph (CG), and Junction Tree (JT).
- The Belief Propagation (BP) inference and decoding technique was analysed.

Implementation

- The Raptor code was applied to the BEC, BSC, and BAWGNC models using FGs.
- The FG, CG, and JT was applied to the Raptor code.
- BP was applied to the PGM to decode the Raptor codes.
- The application of Tree-structure Expectation Propagation (TEP) and Inactivation Decoding (ID), two graph altering derivatives of the BP decoding algorithm, were covered and applied to the Raptor code.

Findings

- The Bit Error Rate (BER) drop-off of Raptor codes using FGs for the BEC, the BSC, and the BAWGNC is similar for each case, indicating the viability of Raptor codes for all 3 channel models.
- The transmission rate of the Raptor code approaches a point close to channel capacity as the number of input symbols $\mathcal{K}$ goes to infinity. That is, a very small overhead is required to achieve reliable communication for large values of $\mathcal{K}$.
- The transmission rate of the Raptor code approaches a point close to the channel capacity as the number of input symbols $\mathcal{K}$ goes to infinity. That is, a very small overhead is required to achieve reliable communication for large values of $\mathcal{K}$.
- The BER of the Raptor code is not limited by an error floor, whereas the LT-code is. This proves that the Raptor code is an improvement on the LT-code.
- Both the ID and TEP improves the BER of the Raptor code when used in conjunction with the BP algorithm. The TEP algorithm is preferable due to its lower computational complexity compared to ID.
- The BER of the CG is better than that of the FG for Raptor codes. The BER of the JT is the best of the 3.

- The computational complexity of the JT exceeds that of the CG, whereas CG again exceeds that of the FG due to the optimised BP algorithms for FGs.

Publications

- Published a paper at Telkom's SATNAC conference of 2012 [15].

1.5 Overview of field and outline of Thesis

In 1948 C.E. Shannon set the basis for achieving reliable communication over noisy channels in his pioneering work *A mathematical theory of communication* [4]. From this channel, models such as the BEC were derived, which emulates a channel that may lose some of the transmitted data. Other models include the BSC and BAWGNC, both of which emulate channels that may induce errors in the data. A vast variety of practical communication mediums can be accurately described by these three channel models. Accordingly we focus on these models in this work and define them in Section 2.2.

Each of these channels has a theoretical limit to how much information it can reliably transmit at any given moment. It is important to determine these limits since they provide a reference against which to measured how close a given transmission algorithm performs to what is possible. These limits are referred to as their *channel capacity* and are covered in Sections 2.3 and 2.4.

As defined in the work of Shannon, these limits depend on the given probability for an error to occur during transmission, yet always remain below the transmission rate $\mathcal{R}$ of 1. Consequently, it is necessary to add redundancy to the original data (from here on referred to as the *source data*) in order to achieve reliable communication over any realisable channel. This is done using error control coding (Section 2.5), of which there are two possible approaches:

- error detection coding [16] and
- Forward Error Correction (FEC)-coding.

As the name suggests, the former can only detect errors, whereas in the case of the latter enough information is available after transmission to detect *and* repair a limited number of possible errors. The number of errors an FEC-code can repair varies by type and design.

This thesis focuses on the decoding of FEC-codes using PGMs, specifically that of Raptor (*rap*id-*tor*nado) codes, which constitute a subset of digital fountain codes. This is a class of codes designed to overcome the limitations of linear block codes with respect to broadcasting while maintaining linearity and low-complexity decoding. Specifically, the addressed limitations are the fixed code rates of block codes and the requirement that all packages be received and be received sequentially. This is covered in Section 2.6.

A popular analogy to explain the main advantage of a fountain code is that of a fountain, hence the name. In this analogy, the fountain is the transmitter, the water drops the transmitted data packets, and the receiver is a bucket next to the fountain. The goal is for the bucket to be filled with water. For this to happen, the bucket only needs to capture enough water droplets. It does not matter which droplets are caught nor in what order. The same applies to fountain codes: it does not matter which packets the receiver obtains, nor whether they are received in order. As long as enough packets are received, the source data can be decoded. This is what makes the fountain code appealing, as these are the typical issues involved with broadcasting applications.

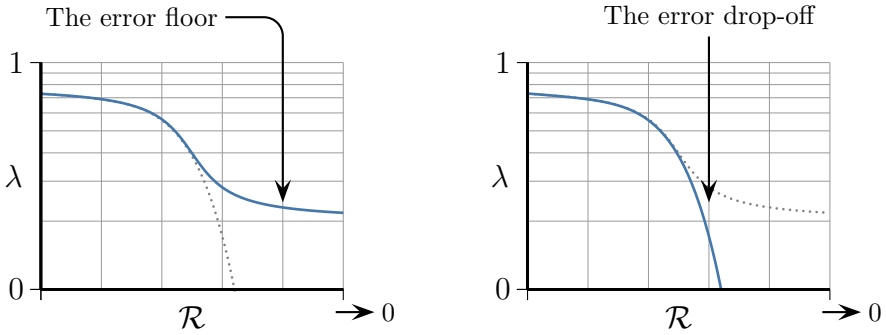
The required $\mathcal{R}$ for communication at channel capacity depends on the error probability of the channel, thus it is typical for an FEC-code to be optimised for a specific error probability. Unfortunately, this FEC-code will consequently perform sub-optimally for any other channel error probability. Fountain codes do not suffer from this limitation when applied to BECs, i.e., fountain codes are able to perform close to the channel capacity for all error probabilities. Thus fountain codes are said to be *universal* for BECs. Unfortunately, fountain codes are not universal for BSCs and BAWGNCs. However, it was found by Omid Etesami et al. [17] that their performance varies only marginally as the error probability is changed, thus remaining suitable candidates for broadcasting applications over these types of channels.

The 1$^{\text{st}}$ theoretical fountain code developed was the random linear fountain code [8, 18]. It encodes the source data packets into transmitted packets uniformly at random, such that each of the encoded packets is a linear combination of a random subset of the source packets. The receiver subsequently makes use of Gaussian Elimination (GE) to recover the source data. This code is universal for BEC, only for a data set of infinite size and its performance diminishes as the data size approaches zero. This, along with the exponential order of complexity of the GE, $O(2^{\mathcal{K}})$, where $\mathcal{K}$ is the data set size, renders

the random linear fountain code impractical.

The 1st practical realisation of a fountain code is the LT-code, which was introduced by M. Luby in [10]. It improves on the random linear fountain code by lowering the complexity to $O(\mathcal{K} \ln \mathcal{K})$ using a specially designed generation distribution to replace the uniform distribution of the random linear fountain, to generate a sparser encoding [8, 18]. The LT-code lowers the complexity further by replacing the GE with an inference algorithm known as the LT-process.

However, the LT-code still suffers from an error floor, which is that the *rate* of decrease of the decoding error probability λ decreases as the transmission rate goes to zero. An arbitrary example of this phenomenon is given in Fig. 1.1. This error floor is a side effect of the distribution the LT-code uses to generate the sparse encoding, which tends to leave a small fraction of the source data unencoded. The receiver will thus never be able to decode these fractions of the source data.



(a) LT-code (b) Raptor code

Figure 1.1: (a) shows the decreasing decoding error probability of an arbitrary LT-code as the transmission rate decreases. The significant decrease in the error drop-off, known as the error floor, can be seen. On the contrary, the Raptor code showns no such decrease, as depicted in (b).

Subsequently, the Raptor code was developed to deal with the fraction of source data missed by the LT-code. It accomplishes this by encoding the source data with a separate sparse-graph code, such as an LDPC-code, which is referred to as the *pre-code*. Thus the transmission data is generated with the LT-code from the data set obtained by encoding the source data with the pre-code.

With most of the source data already encoded by the pre-code, the fountain code is only required for its fountain like properties. This allows for an even sparser generation matrix to be used by the Raptor code than that of the LT-code. Thus the Raptor code uses a distribution that results in a higher fraction of the source data not to be encoded, relying on the pre-code to decode these fractions. This decreases the decoding complexity even further from the *super-linear* complexity growth of $O(\mathcal{K} \ln \mathcal{K})$ of the LT-code to only a *linear* complexity growth of $O(\mathcal{K})$ [18].

Traditionally, FEC-codes are decoded using the BP algorithm on tanner graphs. This approach is designed for deterministic decoding, i.e., choosing between a 1 or a 0 for the value of each bit at each decoding step, based on its neighbouring variable values. It turns out that this approach is a subject of the field of PGMs. This implies that other graph models and decoding techniques may also be applied to these FEC-codes, potentially with improved results.

The investigation of some of these PGMs models and techniques follows in Chapter 3. In Section 3.2 basic graph terminology is covered, such as defining the degree of a node or a cyclic graph, thereby giving the reader the necessary background terminology for the work ahead.

The first two PGMs we consider is the BN in Section 3.3 and the MRF in Section 3.4. These graphs are similar in that they represent the joint Probability Distribution (PD) of the transmitted data as a product of PDs, referred to as *clusters*, and represent each variable as a node. The BN represents these marginals as conditional PDs, where dependencies between nodes is only in 1 direction. Contradictory to this, the MRF represents these marginals as joint PDs where dependencies are mutual. Despite these differences, it is possible to convert a BN to a near equivalent MRF and *vice versa*.

In Section 3.5 we find that, for any BN or MRF, there exists an equivalent FG that expresses the same conditional independencies as well as the same factorisation of the join distribution. This is achieved by adding an extra node for each cluster that contains the PD of that cluster. These are called *factor nodes* and have the purpose of describing the dependencies of the variables in the graph.

However, it is not required for each variable to be uniquely assigned to a node. The CG lumps all the variables pertaining to a certain cluster into a single node, as covered in Section 3.6. This releases the CG from the limit of performing inference over only a single variable at a time, as is the case for BNs,

MRFs, and FGs. Although this improves our ability to do inference, and thus improve the odds for successful decoding, it also increases the size of the PDs that are propagated across the graph. This increases the total computation complexity at an order of $O(2^{\mathcal{N}})$, where $\mathcal{N}$ is the number of variables of the largest cluster in the graph.

Unfortunately, for any practical application, MRFs, FGs, and CGs most often contain loops within their structure. As a result we can only do *approximate* inference on these graphs, as opposed to *exact* inference. There are thus two models of inference that is covered. The first, approximate inference, is applied to MRFs, FGs, and CGs and allows local optima that can only be found using an iterative procedure. This process is not guaranteed to obtain the optimal solution after decoding. The second inference model, exact inference, has a single optimum that can be found in a finite number if steps. This inference model does guarantee an exact solution.

To effect exact inference, a graph with a treelike structure is required, i.e., a graph with no loops. It is possible to construct a CG with a treelike structure for any given PD. Such a graph is known as a Junction Tree (JT), as is covered in Section 3.7. Most CGs have multiple possible equivalent JTs. However, some of these JTs will have lower computational complexities than others. Unfortunately, finding the optimal equivalent JT remains a Non-deterministic Polynomial-time hard (NP-hard) problem.

Once a graph is constructed, each PD is confined to its relating cluster, i.e., each cluster knows nothing of any cluster's likelihoods but its own. The word "*knows*" is used since the PD of a cluster is information. The total of all the clusters' information is equivalent to the information of the original joint PD on which the graph is based. To decode the graph, this information must be propagated across its entirety, giving each cluster access to the whole body of information from which it can calculate its local likelihoods. In Chapter 4 we review how this is done by applying message-passing algorithms such as BP.

We considered the message-passing principles that most of these probabilistic models are based upon in Section 4.2. These principles apply to any graph that has a treelike structure and they have an order of complexity proportional to the number of nodes in the graph. A message-passing algorithm is typically started at the *leaf nodes*, i.e., nodes that only have one other node connected to them via an edge. It terminates once all nodes have transmitted a message to all of its neighbours, at which stage all of the nodes contain all

of the information within the graph.

Furthermore, each node must only receive the information exactly once, otherwise the information will be duplicated and the final result skewed. Unfortunately, in the case of graphs with loops, this is not possible to guarantee and thus it is impossible to guarantee an exact solution as an end result.

Section 4.3 sees the use of the Variable Elimination (VE) algorithm to describe how message-passing applies mathematically to the joint PD of a graph and how it reduces the computational complexity of inference over a graph. It is found that VE algorithm's computational complexity is linearly proportional to the number of variables in the graph, in contrast to the exponential computational complexity scaling of the naive approach.

Section 4.4 gives a qualitative description of how BP extends the concept of message-passing in order to apply inference over FGs. Also known as the sum-product algorithm, it involves only the two operations from which the name is derived, i.e., summations and products. Since there are two types of nodes in a FG, the sum-product involves two types of messages. BP over graphs with loops is known as loopy BP and can only execute approximate inference.

For decoding an FEC-code it is not required to find the true PD, but only the values that maximise it. This allows for the reduction in complexity of the BP algorithm by reducing the marginalisation operations to maximisation operations and operating in the Log-Likelihood Ratio (LLR) domain. This form of the BP algorithm is known as the Viterbi or Min-sum algorithm.

The order in which the messages are passed is referred to as the schedule. There exist many valid schedules, some more efficient that others. In Section 4.5 we only consider three of the more efficient and widely use schedules, namely the

- Standard forward-backward execution,
- Message flooding (synchronous BP), and
- Sequential message-passing (asynchronous BP).

The *Tanh-rule*, in Section 4.6, reduces the complexity of the BP from $O(2^{\mathcal{N}})$ to $O(\mathcal{N})$. However, this approach requires that the messages be confined to only a single Binary Random Variable (BRV), which severely limits its

usefulness as it cannot be applied to CGs, JTs, or any graph with a non-binary data set.

BP over CGs may be considered to be a more generalised form of BP, as explained in Section 4.7. The advantage of this more generalised form of BP lies in the fact that these messages are not restricted to a single variable. This allows for more information to be propagated for every message passed. Thus, on average, a CG or JT will converge in fewer iterations than its equivalent FG, BN, or MRF. Moreover, fewer messages implies that less potential biasing will occur in the case of graphs with cycles.

In Chapter 5, the aforementioned channel models and decoding techniques are applied to the Raptor code. The chapter begins with the general application of PGMs on FEC-codes in Section 5.2. This mainly concerns the values observed at the receiver, which do not pertain to any of the variables included in the original graph of the code and must therefore be added as additional entities. However, by applying exact inference to these new entities, the complexity of the graph may be reduced before running the iterative process of loopy BP.

The standardised Raptor 10 (R10) code is covered next in Section 5.3, and is chosen for its lower complexity as compared to the RaptorQ (RQ) code. The R10 is a systematic code designed to be encoded up to 2^{13} source packets and 2^{16} transmission packets and is also designed specifically for the application of the ID algorithm.

As explained in Section 5.4, decoding any practical Raptor code's FG is done using loopy BP, where messages are iterated across the graph until convergence is achieved, i.e., the amount at which the probabilities in the graph change decreases below a given value or a maximum number of iterations is reached. If the maximum number of iterations is reached, the decoding has failed.

One fundamental problem with fountain codes is that sufficient transmitted packets that only encode a single source packet are required in order to initialise the BP algorithm and to successfully decode the LT-code. We will refer to such a packet as a $\mathcal{D} = 1$ packet, where $\mathcal{D}$ is the number of source packets it encodes.

As explained in Section 5.5, with very little to no increase in complexity, we can use the relationship of the source packets and transmitted packets to change the structure of the FG in order to obtain the necessary $\mathcal{D} = 1$ packets. Thus if BP fails, one of two algorithms may be used in an attempt to improve

the recovery rate: TEP or ID.

TEP is covered in Section 5.5.1, which is proposed and analysed by P.M. Olmos et al. in [19]. The algorithm is based on an observation that the relationship of the graph variables, as defined by the factor nodes, are simultaneous eXclusive-OR (XOR) equations. By manipulating these equations, the algorithm changes the graph structure of an FG, attempting to generate more $\mathcal{D} = 1$ packets from the existing $\mathcal{D} = 2$ packets. Every time a BP decoding attempt fails, the graph alteration process will be repeated until either BP is successful, or no more $\mathcal{D} = 2$ LT-factors remain.

ID, as covered in Section 5.5.2, was developed in order to combine the decoding success rate of GE with the low complexity of BP [20]. The algorithm starts by searching for a $\mathcal{D} = 1$ packet. If found, it performs GE on the LT parity-check matrix, thus removing the source packet encoded by the $\mathcal{D} = 1$ packet from all other factor nodes. This source packet is now considered *recovered*. If no $\mathcal{D} = 1$ packets are left, the algorithm *inactivates* a source packet that has not yet been recovered, i.e., this source packet is no longer included when calculating the value of $\mathcal{D}$ for any packet. Thereafter, another search is done for a $\mathcal{D} = 1$ packet. This continues until all source packets are either recovered or inactivated. Finally, BP is applied to the newly formed FG.

Section 5.6 covers the application of the CG to the Raptor code. These are generated starting only with the pre-code, negating the necessity of reproducing this section of the Raptor code as each transmitted packet is received. Each cluster corresponding to a transmitted packet may then be added on-the-fly without altering the existing graph. Similarly to the FG, a Raptor code CG is decoded using loopy BP.

In Section 5.7 the JT is constructed using the VE algorithm, which will *always* produce a treelike structured graph. The order in which the cluster of the JT is generated is chosen via the *greedy search* algorithm, where the decisions are made *on-the-fly*, i.e., cluster number N is chosen after cluster $N - 1$ and before cluster $N + 1$. The *min-weight* cost function is also used, where the cost of each cluster generated is based on the the size of its PD. The process of decoding a JT is very similar to that of a CG, except that exact inference is done and thus no iteration is required.

Chapter 6 shows the results of the simulations done to test and confirm the topics and theories discussed within this thesis.

Simulations of Raptor code using loopy BP and the FG over the BEC, the BSC, and the BAWGNC were done and their results are given in Section 6.2. The Raptor code used for these simulations is the R10-code with a source data size of 1kb. The error probabilities of the channels were chosen such that their channel capacities were all at $\mathcal{R} = 0.5$. It was found that in all 3 cases the Raptor code has a similar BER drop-off, showing that it is a viable code to use for all 3 channel models.

In Section 6.4 we show that the BER of the Raptor code is not limited by an error floor, whereas the LT-code is. For these simulations the BSC with a channel capacity of $\mathcal{R} = 0.5$ was used and the tanh-rule loopy BP was applied to the FG to decode both the R10- and LT-code. A source data set of 1kb was used. These simulations illustrate that the Raptor code is an improvement on the LT-code.

The graph altering algorithms, TEP and ID, were tested on the Raptor code and compared against the tanh-rule loopy BP algorithm, for which the results may be found in Section 6.5. These test simulations were run over the BSC, using FGs to decode the R10-code with a source data set of 1kb. It was found that both the ID and TEP algorithms showed improvements in their BER with respect to the tanh-rule loopy BP algorithm. The TEP is considered preferable due to its lower computational complexity compared to that of ID.

The BERs of the FG, CG, and JT when applied to a Raptor code are compared in Section 6.6. These simulations were done at the channel capacity of $\mathcal{R} = 0.5$. A Raptor code with an LDPC pre-code and a data set of 100 bytes were used. In Fig. 6.5 it can be seen that the BER of the CG is better overall than that of the FG, and that of the JT is the best of the 3.

The computational complexities of the FG, CG, and JT when applied to a Raptor code are given in Table 6.2. This was done by comparing the number of messages passed before successful decoding was achieved. These simulations were done at the channel capacity of $\mathcal{R} = 0.5$. A Raptor code with an LDPC pre-code and a data set of 100 bytes were used. It was found that the computational complexity of the JT exceeds that of the CG, where the CG exceeds that of the FG.

Finally we conclude in Chapter 7 that the Raptor code is an excellent solution for broadcasting applications over not only the BEC, but also the BSC and BAWGNC. Furthermore, even though the CG and JT BER curves

show significant improvement over that of the FG, unfortunately their computational complexities may inhibit their practical use for current generating architectures. It was also concluded that the TEP is an improvement over the BP algorithm when decoding Raptor codes using FGs.

Chapter 2

Information theory & Error control coding

2.1 Introduction

All forms of communication are in essence the transmission of data from one point in space to another, and all realisable forms of communication are susceptible to noise. This presents a problem, as the noise may corrupt any transmitted data, making such communication unreliable. The purpose of information theory is to achieve reliable communication when noise is present. This chapter will focus on developing an understanding of how this is achieved and the background theory of the mechanisms used.

In 1948, C.E. Shannon set the basis for approaching this communication problem in his pioneering work *A mathematical theory of communication* [4] where a schematic diagram of a general communication system similar to Fig. 2.1 was presented. We will consider the topics discussed in this chapter as we elaborate on this figure.

The system in Fig. 2.1 consists of six distinct parts. First we have the *source* containing the data we desire to communicate. For the scope of this thesis we will limit this and all other data to *binary* data. This data is grouped into *packets* that are passed on to an *encoder*. The encoder then *encodes* these packets, thereby adding *redundancy* to the source data, as well as converting them into a signal suitable for transmission. This is covered in Section 2.5.

After encoding, the packets are transmitted. The medium through which the data is transferred is called the *channel*, which delivers the packets to the

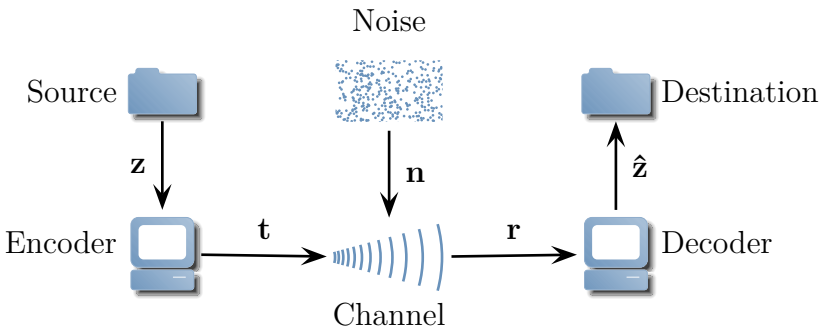


Figure 2.1: Schematic of a general communication system: The source passes the packet $\mathbf{z}$ to an encoder, which adds some redundancy to obtain the transmitted message $\mathbf{t}$. The channel adds some noise $\mathbf{n}$ to $\mathbf{t}$, yielding a received message $\mathbf{r} = \mathbf{t}+\mathbf{n}$. The decoder uses the known redundancy to infer both the original signal $\hat{\mathbf{z}}$ and the added noise $\mathbf{n}$.

decoder in a sub-optimal form due to the addition of noise to the packages. The result is a reduction of our confidence in the received data. In Section 2.2 we will consider the channel models Binary Erasure Channel (BEC), Binary Symmetric Channel (BSC), and Binary Additive White Gaussian Noise Channel (BAWGNC); three fundamental ways in which the effects of noise are modelled. The section following this (Section 2.3) covers the topic of how information can be quantified. In Section 2.4 we then use these measures to define the limitations of the channel models named above.

Finally, the *decoder* performs the inverse operation of that done by the encoder, reconstructing the original message from the received signal. This is done using the known redundancy introduced by the encoding system to infer both the original data as well as the added noise. This is briefly touched on in Section 2.5. The application of fountain codes for decoding is covered in Section 2.6 in more detail.

2.2 Channel models

As mentioned before, the medium through which the data is transferred is referred to as the *channel*, which induces noise into the carrier signal. Before we continue with the details of how we encode and decode our data, we need to model the manner in which these noisy channels corrupt the data.

A channel could exist in many forms, e.g., wired vs wireless channels, all of which have different physical attributes. In order to maintain an accurate

representation of these channels, some are modelled differently to others. The models we focus on in this thesis are the BEC, BSC, and BAWGNC. These three models are sufficiently generalised to model most real world binary channels accurately, whether individually or as a combination.

These channel models are all Binary Input Memoryless Symmetric Channels (BIMSCs), of which a general depiction is given in Fig. 2.2 [21]. They consist of an input set $\mathcal{A}_t = \{0, 1\}$, an output set $\mathcal{A}_r = \{a_0, a_1, \ldots, a_{\mathcal{B}-1}, a_{\mathcal{B}}\}$, transition probabilities $\mathcal{P}_t = \{p_0, p_1, \ldots, p_{\mathcal{B}-1}, p_{\mathcal{B}}\}$, and a number of outputs $\mathcal{B}$. Set $\mathcal{P}_t$ dictates the probability of any transmitted bit t resulting in an output r, e.g., in Fig. 2.2 the probability of $t = 0$ resulting in $r = a_1$ is p_1.

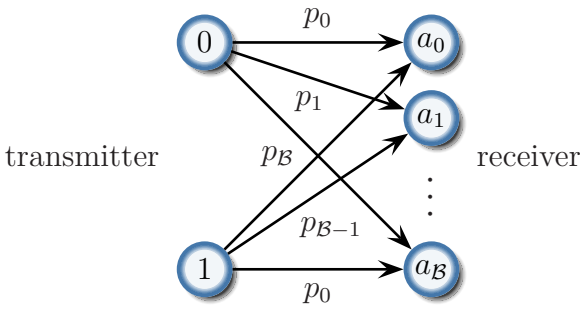


Figure 2.2: A general BIMSC with the input set $\mathcal{A}_t = \{0, 1\}$ (left), output set $\mathcal{A}_r = \{a_0, a_1, \ldots, a_{\mathcal{B}-1}, a_{\mathcal{B}}\}$ (right), and the flow of data acting from left to right. The transition probabilities are given in table 2.1.

These channels are described as "memoryless" since their inputs and outputs at time $T = t$ are independent of all other inputs and outputs at time $T \neq t$, as defined in [3]. The symmetric property of the channels relates to their transition probabilities. For binary input channels, this implies that the order of the transition probabilities of input $t = 1$ is in reverse order to that of $t = 0$[1]. As an example the transition probabilities of Fig. 2.2 are given in Table 2.1.

Table 2.1: The transition probabilities of the BIMSC in Fig. 2.2.

	$r = 0$	$r = 1$	$\ldots$	$r = \mathcal{B} - 1$	$r = \mathcal{B}$
$t = 0$	p_0	p_1	$\ldots$	$p_{\mathcal{B}-1}$	$p_{\mathcal{B}}$
$t = 1$	$p_{\mathcal{B}}$	$p_{\mathcal{B}-1}$	$\ldots$	p_1	p_0

[1] A more general definition of a symmetrical channel, which encompasses non-binary channels as well, is given in [21].

2.2.1 Binary Erasure Channel

The BEC is used to describe a channel were packets are either received uncorrupted and are therefore completely reliable, or not received at all. The phenomenon of losing transmitted packets is referred to as *packet dropping*, where the lost packets themselves are called *erasures*.

Fig. 2.3 depicts this channel model as a BIMSC with $\mathcal{B} = 3$. An erasure is indicated with the question mark symbol and the probability of an erasure occurring is defined by the constant ϵ. From this figure we see that when a bit is received, we are absolutely certain it is received correctly. For example, consider the case where a 0 is transmitted. The probability of the receiver receiving a 0 is $P(0) = 1-\epsilon$ and the probability of the bit being lost is $P(?) = \epsilon$, i.e., the probability of receiving a 1 in this case is $P(1) = 1 - P(0) - P(?) = 0$.

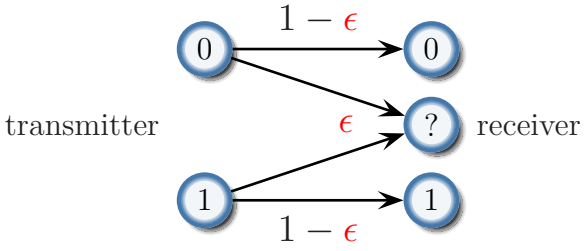


Figure 2.3: The BEC with erasure probability ϵ. The channel has 0 and 1 as possible inputs (left side). The channel output (right side) has 3 possible outcomes: 0, 1, and an erasure represented by the symbol ?. The probabilities of these outcomes depend on the original input and ϵ.

There exist very few real world applications for this model, although one application is significant enough to make the BEC model worth mentioning, namely the internet. Due to the existing architecture of the internet, most packets are received correctly. However, due to router flooding it is possible that packets may be dropped, causing an erasure. It is, i.a., for this reason that the Raptor code was originally designed for BEC [8]. However, in this thesis we consider the Raptor code for use over BSCs and BAWGNCs, thus the BEC applies as a reference for the work ahead.

2.2.2 Binary Symmetric Channel

Unlike the BEC, the BSC does not allow for erasures, but rather accepts all transmitted data with a fixed error probability. This opens the possibility to

interpreting some of the data incorrectly, e.g., interpreting a transmitted 0 as a 1. These misinterpretations are generally referred to as *bit flips*. Figure 2.4 represents a BSC with the constant bit flip probability ρ.

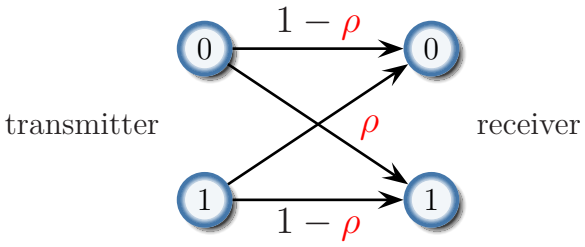


Figure 2.4: The BSC with bit-flip probability p. The channel has 0 and 1 as possible inputs (left side) and outputs (right side), i.e., $\mathcal{B} = 2$.

As we can see, e.g., if we receive a 1, we cannot definitely say that a 1 was transmitted and thus we need to assign a Probability Distribution (PD) to what we *believe* was transmitted. Since there are only two possible inputs, the PD is defined by the Bernoulli function:

$$P(t|r) = \rho^{|t-r|}(1-\rho)^{1-|t-r|} \tag{2.2.1}$$

where r is the observed outcome and t is the transmitted bit.

Even though the BSC is more common in practice than the BEC, it does have one drawback: a fixed value for the bit flip probability. That is, the BSC model assumes a common Signal-to-Noise Ratio (SNR) for all packets received at every interval of time. If the SNR varies over time, ρ can only be approximate using the average SNR. To manage such a scenario more efficiently, another model may be used that assigns a bit flip probability to each packet based on the SNR at the time it is received. This model is known as the BAWGNC.

2.2.3 Binary Additive White Gaussian Noise Channel

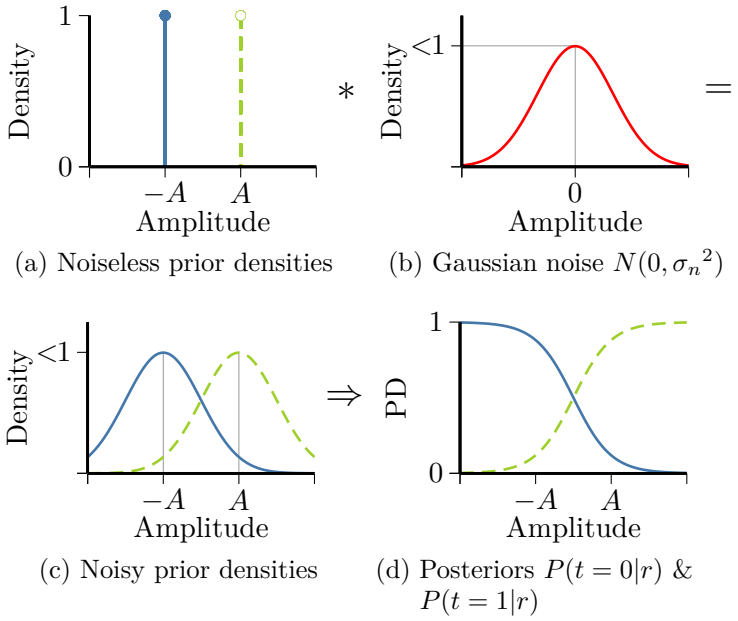
In practice, the transmitted signal is exposed to many types of noise from different sources that are additive and independent of one another. According to the the central limit theorem, these noise sources may be approximated by a Gaussian random variable $\mathbf{n}$.

The output of the BAWGNC is defined as $\mathbf{r} = \mathbf{t} + \mathbf{n}$, where $\mathbf{t} \in \mathcal{A}_t = \{0, 1\}$ is the channel input and $\mathbf{n}$ is a zero-mean Gaussian random variable with

variance ${\sigma_n}^2$. The Gaussian Probability Density Function (PDF), with mean μ and variance ${\sigma_n}^2$, is denoted as follows:

$$N(\mu, {\sigma_n}^2) \stackrel{\text{def}}{=} \frac{1}{\sqrt{2\pi{\sigma_n}^2}} e^{-\frac{(r-\mu)^2}{2{\sigma_n}^2}}.$$

Since this is a continuous function, the model has a continuous channel output set $\mathbf{r} \in \mathcal{A}_r = (-\infty, \infty)$. This implies that $\mathcal{B} = \infty$; nonetheless, it adheres to the properties of a BIMSC.



(a) Noiseless prior densities (b) Gaussian noise $N(0, {\sigma_n}^2)$

(c) Noisy prior densities (d) Posteriors $P(t=0|r)$ & $P(t=1|r)$

Figure 2.5: The behaviour of the BAWGNC. In (a) we see the PD at the transmitter for either of the given inputs 0 or 1. (b) shows the additive Gaussian noise induced on the inputs, and (c) is the resulting output distributions. By using the Bayes' theorem in (2.2.2), we obtain the posterior PDs in (d).

Figure 2.5 illustrates the effect the BAWGNC has on the transmitted data, given that a bipolar amplitude modulated signal is used. This type of modulation translates the channel input set $\mathcal{A}_t = \{0, 1\}$ to the signal amplitude set $\mathcal{A}_a = \{-A, A\}$. The solid lines (blue) show the change in the PDF of $P(t = -A|r)$ in Figs. 2.5(a) through 2.5(c), as well as its final posterior PD in Fig. 2.5(d). Likewise, the dashed (green) lines show the same for $P(t = A|r)$ throughout.

Were we to transmit our signal across an ideal channel with no noise, the posterior probability densities would be as shown in figure Fig. 2.5(a).

These densities state that any signal transmitted will be received exactly the same, with no ambiguity. The addition of noise to the signal is described by the correlation of these densities with that of the Gaussian noise shown in Fig. 2.5(b). This results in the noisy prior densities in Fig. 2.5(c).

To discern the probability of the original inputs from each outcome, we make use of Bayes' theorem:

$$P(t|r) = \frac{P(r|t)P(t)}{\sum_t P(r|t)P(t)} \tag{2.2.2}$$

where r is the observed outcome[2] and t is the transmitted bit. Given that the noise is Gaussian, we have the following probabilities:

$$P(r|t=0) \sim N(-A, {\sigma_n}^2)$$
$$P(r|t=1) \sim N(+A, {\sigma_n}^2).$$

Furthermore, it is reasonable to postulate that for any large quantity of transmissions, the number of each value in $\mathcal{A}_t$ transmitted is equal [16], i.e., $P(t=-A) = P(t=A) = 0.5$. The PD of the channel input given the channel output is

$$\begin{aligned} P(t|r) &= \frac{N(t, {\sigma_n}^2)}{N(-A, {\sigma_n}^2) + N(A, {\sigma_n}^2)} && \text{for } t \in \{-A, A\} \\ &= \frac{\exp\left(\frac{(r-t)^2}{{\sigma_n}^2}\right)}{\exp\left(\frac{(r-A)^2}{{\sigma_n}^2}\right) + \exp\left(\frac{(r+A)^2}{{\sigma_n}^2}\right)} && \text{for } t \in \{-A, A\} \\ &= \left(1 + \exp\left(\frac{2rt}{{\sigma_n}^2}\right)\right)^{-1} && \text{for } t \in \{-A, A\}. \end{aligned}$$

It is possible to approximate a channel with BAWGNC properties as a BSC. To do this, a fixed threshold is assigned in the centre between the two means, as indicated in Fig. 2.6. Each bit we receive is then assessed against it. If the observed outcome is more than the threshold, the PD $P(t|r=1)$ is assigned using the Bernoulli function (2.2.1), otherwise we assign $P(t|r=0)$. The bit flip probability ρ may be calculated by integrating over the area of the probability density of $P(t=-A|r) >$ threshold or $P(t=A|r) <$ threshold.

The original PD assignment using Bayes' theorem is known as the soft-assignment of the bit, whereas the BSC approximation is known as the hard-assignment [22]. The hard-assignment is often used in practice to conserve energy and computational power, especially for real-time applications. However,

[2] A sample from one of the distributions in Fig. 2.5(c)

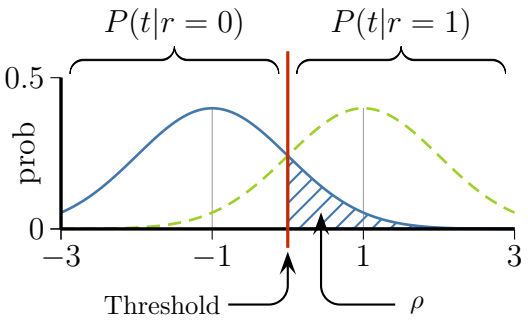


Figure 2.6: The approximation of the BAWGNC as a BSC. The denominator between the two possible PDs of what the outcome may present (the threshold) is set at the centre between the two means. Whether the observed outcome is below or above the threshold will determine the PD used. Since the approximation fixes the error probability, we lose information.

in doing so we lose a great deal of information by replacing the true likelihood with a fixed value. For example, in Fig. 2.6, $P(t = A|r = 3) >> P(t = A|r = 0.5)$ when using soft-assignment, however, $P(t = A|r = 3) = P(t = A|r = 0.5)$ with hard-assignment. This approximation of the received data reduces the channel capacity, as is discussed in Section 2.4.

2.3 The measure of information

In order to determine the efficiency of any encoding and decoding algorithm, we need to know what the optimal achievable performance over the given channel is. That is, what is the theoretical limit to how much information we can transmit and decode without errors over the given channel at any given moment. Since only *binary* data is considered in this thesis, it follows that what must be defined is some measure of the amount of information *each bit* contains in order to establish these theoretical limits.

For this task we state the following formal definition of a Discrete Random

Variable (DRV), called an ensemble:

Definition 2.1: An ensemble [2, p. 22]
An ensemble X is a triple $(x, \mathcal{A}_X, \mathcal{P}_X)$, where x is an DRV with cardinality $|X|$. The variable x is defined by a set of possible values $\mathcal{A}_X = \{a_1, a_2, \ldots, a_i, \ldots, a_{|X|}\}$, having the respective probabilities

$$\mathcal{P}_X = \{p_1, p_2, \ldots, p_i, \ldots, p_{|X|}\},$$

given that $P(x = a_i) = p_i$, $p_i \geq 0$ and $\sum_{i=1}^{|X|} p_i = 1$.

For the cases where x is a Binary Random Variable (BRV), the value set $\mathcal{A}_X$ may have more than 2 outcomes, yet is always defined in base 2. For example, if $|X| = 4$ we have $\mathcal{A}_X = \{00, 01, 10, 11\}$. Note that we may define other ensembles, such as $Y \equiv (y, \mathcal{A}_Y, \mathcal{P}_Y)$, through the course of this thesis.

We now consider the work of C. Shannon to provide a measure of information with the logarithmic measures called *information content* and *entropy* [2, 4].

2.3.1 Shannon's information content

Given ensemble X, the information content of the outcome a_i is

$$h(x = a_i) \stackrel{\text{def}}{=} \log_2 \frac{1}{P(x = a_i)}.$$

This choice of a logarithmic inverse function of $P(x)$ is fully explained in [4]. In short, it fits well with our intuition of the proper measure of information as well as its practical and mathematical suitability for the task.

The choice of base 2 logarithm is for the sake of convenience and is based on the focus of this thesis on the binary domain. However, the base may be any number as long as it remains constant throughout. In base 2, this measure indicates the equivalent number of bits[3] worth of information gained by learning the outcome of a BRV.

An important property of the measure is that it is an inverse proportional function to the probability of the outcome. This implies that the less likely an

[3]This may also be some fractional number, e.g., 2.34 bits'

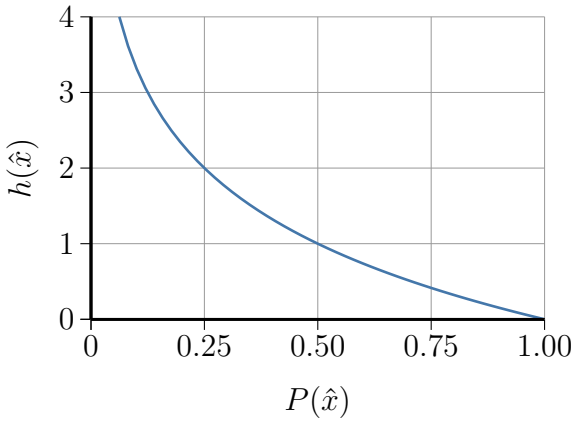


Figure 2.7: This graph shows the Shannon information content function against the entire range (0,1) of possible probabilities for an outcome $\hat{x}$ of the DRV x. Note that at $P(\hat{x}) = 1$ the information content is 0. This agrees with the fact that, because we know the outcome beforehand, we gain nothing in receiving it.

outcome, the more information it delivers, which is evident from the illustration in Fig. 2.7. To elaborate, when the probability of $x = a_i$ is 100%, we gain no information when it is received, as we already knew the result beforehand. However, as the probability decreases, the information content of that outcome increases exponentially.

An example use by MacKay in [2, p. 22] illustrates this concept: take the letters 'e' and 'z' in the English language with respective frequencies of 12.70% and 0.074% according to [23]. With these frequencies, the outcome $x = \mathsf{e}$ in any English document has an information content of 2.98 bits where $x = \mathsf{z}$ has an information content of 7.08 bits. This makes sense if we realise that the observation of the letter 'z' in a word narrows the number of possible words far more than the presence of the letter 'e'.

2.3.2 Entropy

Marginal entropy The concept of information content may be extended such that we may measure the *average* information content of an DRV x. This is called the *entropy* of the ensemble X and is defined as

$$H(X) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{A}_X} P(x) \log_{|X|} \frac{1}{P(x)}, \tag{2.3.1}$$

where $|X|$ is the cardinality of X and $\log_{|X|}\alpha$ is the base $|X|$ logarithm of the variable α. We can also refer to the entropy of an ensemble as the *marginal*

entropy or the *uncertainty* of its outcomes. This is due to the direct correlation of the information content of an outcome and the inverse of its probability. A few properties of $H(X)$ are

- $H(X) \equiv 0$ for $P(x) = 0$ since $\lim_{\varepsilon} \varepsilon \log_{|X|} \frac{1}{\varepsilon} = 0$.
- $0 \leq H(X) \leq 1$
- It is a maximum when $\mathcal{P}_X$ is uniformly distributed.

Because of our use of binary channels, we are specifically interested in an entropy function used often enough to be known as the *binary entropy function*. This function is denoted with the subscript 2 and is defined as follows:

$$H_2(\rho) \stackrel{\text{def}}{=} \rho \log_2 \frac{1}{\rho} + (1-\rho) \log_2 \frac{1}{1-\rho}.$$

As the name suggests, this function describes the entropy of an ensemble with only 2 possible outcomes, i.e., $\mathcal{A}_X = \{0, 1\}$ and $\mathcal{P}_X = \{\rho, 1-\rho\}$. Figure 2.8 shows the binary entropy function over the entire range $\rho \in [0, 1]$. Note that, similar to the first property mentioned above, $H_2(X) \equiv 0$ for $\rho = 0, 1$.

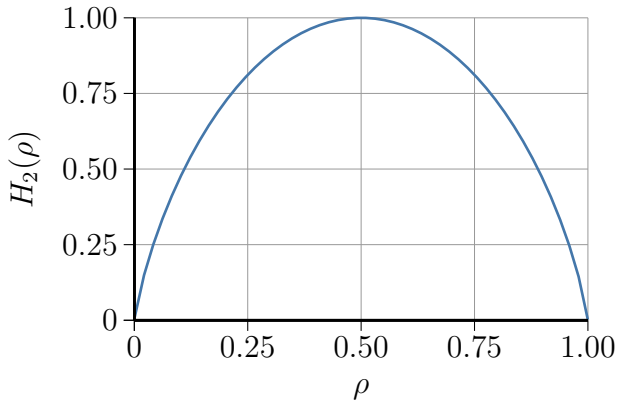


Figure 2.8: The binary entropy function over the range of $\rho \in [0, 1]$. Note that the function is a maximum where $\mathcal{P}_X$ is uniform, i.e., $\rho = 0.5$.

Conditional entropy Given that ensemble X is dependent on the ensemble Y, we may express its entropy simply by writing (2.3.1) as a dependant of observed outcome y:

$$H(X|y) = \sum_{x \in \mathcal{A}_X} P(x|y) \log_{|X|} \frac{1}{P(x|y)}.$$

This function may be extended as a function of the whole ensemble Y, such that

$$\begin{aligned} H(X|Y) &\stackrel{\text{def}}{=} \sum_{y\in\mathcal{A}_Y} P(y)H(X|y) \\ &= \sum_{x\in\mathcal{A}_X}\sum_{y\in\mathcal{A}_Y} P(x,y)\log_{|X|}\frac{1}{P(x|y)}. \end{aligned} \tag{2.3.2}$$

Since X is conditional to Y and the outcome y is assumed to be known, it follows that

$$\begin{aligned} H(X|Y) &= H(X) \quad \text{iff } P(x,y) = P(x)P(y) \\ &< H(X) \quad \text{otherwise.} \end{aligned} \tag{2.3.3}$$

Joint entropy It is also possible to describe the joint entropy of multiple ensembles. For the ensembles X and Y, we may describe their joint entropy as follows:

$$H(X,Y) \stackrel{\text{def}}{=} \sum_{x\in\mathcal{A}_X}\sum_{y\in\mathcal{A}_Y} P(x,y)\log_{|X|}\frac{1}{P(x,y)}.$$

It is proven in [2, 24] that entropy is additive for independent random variables, thus

$$\begin{aligned} H(X,Y) &= H(X) + H(Y) \quad \text{iff } P(x,y) = P(x)P(y) \\ &< H(X) + H(Y) \quad \text{otherwise.} \end{aligned}$$

Mutual information If X and Y are not independent, then learning about the one will give us some information about the other. The average amount of information x contains about y, or *vice versa*, is known as the *mutual information* of X and Y and can be calculated as

$$I(X;Y) \stackrel{\text{def}}{=} H(X) - H(X|Y). \tag{2.3.4}$$

Expressed in words: the information gained about X by observing Y is the total obtainable information of X, less the information of X not gained by observing Y. The mutual information is commutative, thus $I(X;Y) = I(Y;X)$.

Venn diagram A convenient way to illustrate the concept of entropy is by means of a Venn diagram as shown in Fig. 2.9. Each circle represents the entropy of its respective ensemble, while any overlapping area denotes their mutual information.[4]

[4]Note that here the Venn diagram indicates information, not sets as per usual [25].

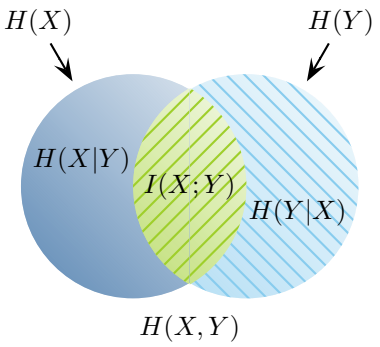


Figure 2.9: A Venn diagram of the field of the collective entropy of ensembles X and Y. The relationships between marginal entropy, joint entropy, conditional entropy, and mutual information are depicted.

2.4 Channel capacity

Given the ability to measure the rate at which information is transferred, we may calculate the maximum achievable rate. This will give us a benchmark to which we may compare the efficiencies of different encoding and decoding algorithms.

The term 'rate of transmission' implies the amount of information about the input of a channel that is gained by the observation of its output. In other words, given that ensemble X and Y define the input and output respectively, what is the mutual information of these 2 ensembles?

This concept is depicted by Berger's entropy diagram in Fig. 2.10: an adaptation of the Venn diagram to represent the transferral of information across a channel. $H(X)$ represents the mean information emitted by the source and $H(Y)$ the mean observed information by the receiver. The increase in uncertainty due to simplifications or omissions during transmission is known as the equivocation, i.e., the uncertainty of X after Y is observed. The nonsensible "information" added to $H(Y)$ by the noise is referred to as irrelevance, i.e., the uncertainty over Y if X is known. The mutual information is the information of the input contained in the output.

From this we may derive that the 'rate of transmission' $\mathcal{R}$ is

$$\mathcal{R} \stackrel{\text{def}}{=} I(X;Y).$$

Causality dictates that the output variable Y is dependent on the input variable X and the channel error probability. Assuming the channel error probability is constant, this implies that $\mathcal{R}$ can be maximised with respect to $\mathcal{P}_X$.

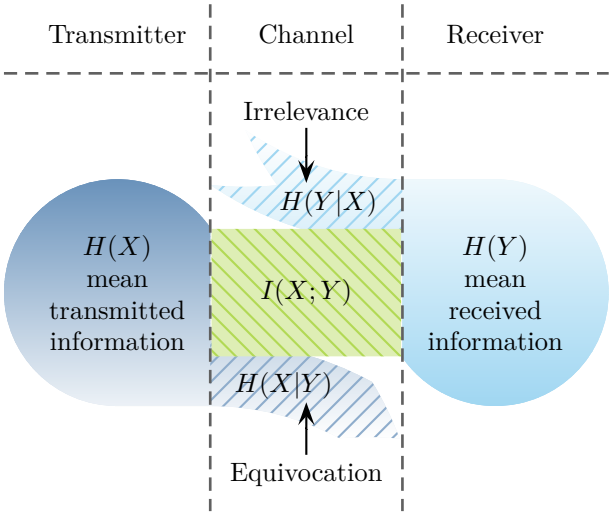


Figure 2.10: Berger's entropy diagram [26] of the same field of the collective entropy as Fig. 2.9. The diagram shows the relationship of the entropy at the input X and output Y of a noisy channel. Irrelevance is added to $H(Y)$ by the channel noise and equivocation is caused due to simplifications or omissions during transmission.

Furthermore, it has been proven by Shannon in [4] that, as long as the actual transmission rate is less than $\mathcal{R}$, *reliable* transmission is achievable, i.e., communication with arbitrary small error.

Moreover, Shannon provided the following theorem:

Theorem 2.1: Shannon's noisy-channel coding [18, p. 152]
Let a discrete memoryless channel $\mathcal{C}$ have the capacity $Cap(\mathcal{C})$, a transmission rate $\mathcal{R}$, and an error probability $\varepsilon > 0$. If $\mathcal{R} \leq Cap(\mathcal{C})$ there exists an encoding algorithm with rate $\mathcal{R}$ such that reliable communication over $\mathcal{C}$ can be achieved with an arbitrary small ε.

Proof: For the proof of this theorem the reader is referred to [4].

Accordingly, the channel capacity is defined as

$$Cap(\mathcal{C}) \stackrel{\text{def}}{=} \max_{\mathcal{P}_X}(\mathcal{R}) = \max_{\mathcal{P}_X}\Big(I(X,Y)\Big). \tag{2.4.1}$$

$\mathcal{P}_X$ maximise $I(X;Y)$ when we have the least possible prior knowledge of the inputs at the receiver. This allows the output to give us the maximum

possible information of the input when receiving it. For a symmetrical channel this is when $\mathcal{P}_X$ is uniform, since no assumption can be made of the input based on this PD. This implies that, for the BIMSC, we have $\mathcal{P}_X = \{0.5, 0.5\}$.

Using (2.4.1) we can establish the channel capacities of the first two models discussed in Section 2.2.

Theorem 2.2: [2, p. 158]
Given the erasure probability ϵ, the capacity of the BEC is $1 - \epsilon$.

Proof: The proof may be found in Appendix A.1.

The BEC capacity above is as expected, since ϵ indicates the average portion of packets that is expected to be dropped, thus the average portion of data transmitted, but not received. The graph of $Cap(\mathcal{C}_{BEC})$ vs. the Bit Error Rate (BER) (ϵ) is shown in Fig. 2.11.

For the BSC we have

Theorem 2.3: [2, p. 158]
Given the bit flip probability ρ, the capacity of the BSC is given by

$$Cap(\mathcal{C}_{BSC}) = 1 - H_2(\rho). \tag{2.4.2}$$

Proof: The proof may be found in Appendix A.2.

The capacity of the BSC vs. BER (ρ) is depicted in Fig. 2.11 as well. The portion in the range $\rho \in [0.5, 1]$ may be counter intuitive. However, this becomes a logical result once it is realised that, for the second half of the range, the information content of the flipped and unflipped bits are reversed. Consider the extreme case of $\rho = 1$; here all transmitted bits are flipped. Thus, all that needs to be done is to flip all the received bits in order to obtain the original package.

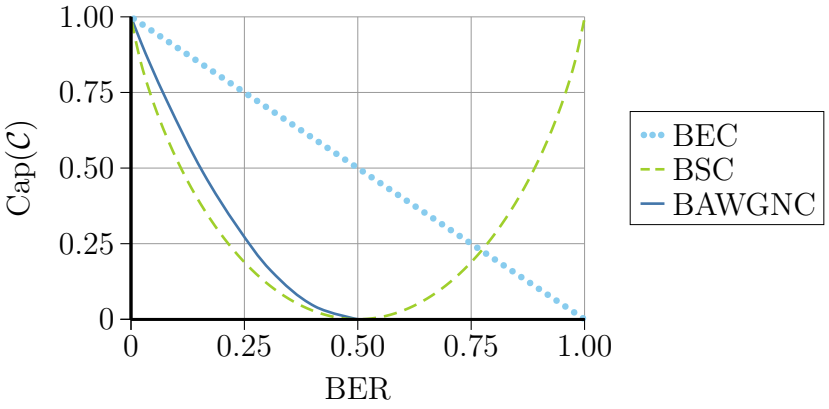


Figure 2.11: The capacities of the BEC, BSC and BAWGNC, where their respective BERs are ϵ, ρ, and $Q({\sigma_n}^{-1})$ as given in (2.4.3). The reflection of $\text{Cap}(\mathcal{C}_{BSC})$ around $\rho = 0.5$ may be attributed to the reversal of information content of the flipped and unflipped bits. $\text{Cap}(\mathcal{C}_{BAWGNC})$ is limited to $\in [0, 0.5]$, since $Q(\infty) = 0$ and $Q(0) = 0.5$.

The equation of the capacity for the BAWGNC is considerably more extensive than the previous two channels.

Theorem 2.4: [16]
Given the noise variance ${\sigma_n}^2$ and the Signal-to-Noise Ratio (SNR) $\frac{A}{{\sigma_n}^2}$, the capacity of the BAWGNC is as follows:

$$Cap(\mathcal{C}_{BAWGNC}) = -\int_{-\infty}^{\infty} g(y, A, {\sigma_n}^2)\log_2\Big(g(y, A, {\sigma_n}^2)\Big)\mathrm{d}y - \frac{1}{2}\log_2(2\pi e{\sigma_n}^2)$$

where

$$g(y, A, {\sigma_n}^2) = \frac{1}{2}\frac{1}{\sqrt{2\pi{\sigma_n}^2}}\left(\exp\left(-\frac{(y-A)^2}{2{\sigma_n}^2}\right) + \exp\left(-\frac{(y+A)^2}{2{\sigma_n}^2}\right)\right).$$

Proof: The proof may be found in Appendix A.3.

$Cap(\mathcal{C}_{BAWGNC})$ is depicted in Fig. 2.11 for the BER and in Fig. 2.12 against the SNR. It is assumed that the energy difference between the two possible outcomes as 1, thus $\text{SNR} = E_N/{\sigma_n}^2 = 1/{\sigma_n}^2$. Also shown if Fig. 2.12 is the approximation of the BAWGNC as a BSC, relating to Fig. 2.6.

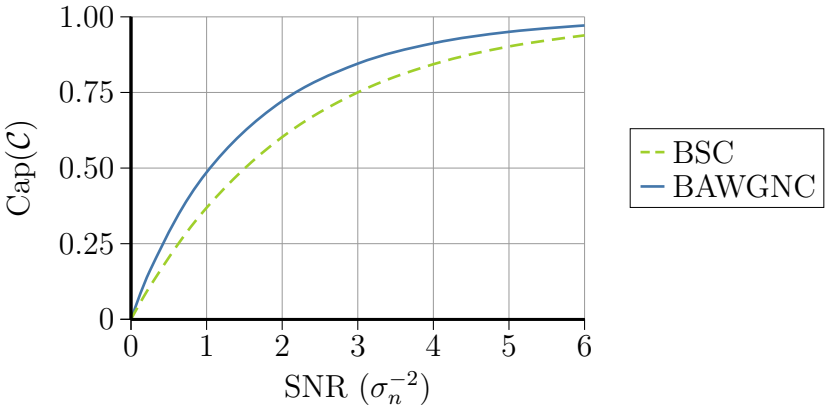


Figure 2.12: The capacity of the BAWGNC against the SNR and approximated BSC capacity. $\text{Cap}(\mathcal{C}_{BSC})$ is calculated using (2.4.2) with $\rho = Q(1/\sigma_n)$.

Harold P. E. Stern et al. shows in [27] that the conversion between BER and SNR is done using the Q-function defined as:

$$Q(x) \stackrel{\text{def}}{=} \int_x^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{y^2}{2}\right) \mathrm{d}y. \tag{2.4.3}$$

and that, for the approximation of the BAWGNC as a BSC, the bit flip probability is $\rho = Q(\sqrt{E_N/2{\sigma_n}^2}) = Q({\sigma_n}^{-1})$. From this we see that, due to the approximation, some information is lost and the capacity reduced. Also note that the maximum BER of the BAWGNC is 0.5, which is equivalent to an infinite SNR.

2.5 Error control coding

The previous section shows that in order to transmit data reliably, we need to add some redundancy to the source data. For example, if we transmit 10 bits of data over BEC with erasure probability $\epsilon = 0.5$, on average we will need to transmit 20 bits in order to convey all the information. Each source data packet and its related redundant data is collectively known as a *codeword*. How we format and manage this redundancy in general is referred to as *error control coding* and is the focus of this section.

There are two possible approaches to error control coding. In the case where we have a duplex communication system, i.e., a system where communication may occur in two directions, we might consider it only necessary to detect an error. Thus, if the receiver detects an error, a request for retransmission of that

codeword may be sent back to the transmitter. This will allow the transmission of codewords with less redundancy. This approach is known as *error detection coding* [16]. One of the most common examples of this approach is the cyclic redundancy check code.

Due to the low redundancy in the codewords of error detection, it may seem that these types of code are capable of communication above the channel capacity. However, this is not the case. Indeed, in the isolated instance where no errors are detected, this is the case. The capacity of a channel is defined over the *average* performance, and in many cases the retransmission of packets is necessary. It is in this that the inevitable redundancy exists that results in a communication rate below capacity.

The second error control coding approach is known as *Forward Error Correction (FEC)-coding*, where enough information is available after transmission to detect *and* repair possible errors. It is thus self-evident that FEC will contain more redundancy in the codewords than error detection. This form of coding is preferable for instance where communication is possible only in one direction, thus not allowing for a request for retransmission.

FEC coding involves two main categories, namely *convolution codes* and *linear block codes* [28]. Convolution codes encode the source data in a bit-by-bit manner by means of linear-feedback shift-registers. These encoded bits are then delivered to the receiver as a continuous stream of bits of a predetermined length. In this thesis we will not make use of convolution codes, thus for more information the reader is referred to D.J.C. MacKay [2] for a basic introduction and T.K. Moon [16] for a more detailed analysis.

Linear block codes segment the original data into packets of equal length. Each packet is then encoded individually into a codeword, here also known as a *block*. These blocks are subsequently passed to the receiver where they are decoded using some decoding mechanism.

For the remainder of this section we will discuss these linear block codes in more detail, since linear block codes are often used as part of the Raptor code. We will start with a more general view of linear block codes with binary inputs, thereafter realising those concepts with the Hamming code. Finally we will consider the more advanced case of Low-Density Parity-Check (LDPC)-codes.

2.5.1 Linear block codes

In their most general form, linear block codes divide the source data into blocks of size $\mathcal{K}$ symbols. However, as our scope is limited to binary channels only, we will limit our consideration of linear block code to that of binary codes. Thus, for our purposes, each symbol consists of a single bit, which implies that each block has $2^{\mathcal{K}}$ possible codewords. We may consider each codeword as an ensemble as defined in Definition 2.1 with a cardinality of $\mathcal{K}$ and $\mathcal{P}$ uniform. For example, a block of size $\mathcal{K} = 2$ has the set $\mathcal{A} = \{00, 01, 10, 11\}$ as possible codewords with probabilities $\mathcal{P} = \{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\}$.

Each block is encoded such that it forms a codeword of size $\mathcal{N}$. By popular notation the shorthand $(\mathcal{N}, \mathcal{K})$ is used before the name of a block code to indicate its encoding behaviour (for example: the (7,4) Hamming code).

Thus, an $(\mathcal{N}, \mathcal{K})$ block code has a code rate $\mathcal{R} = \mathcal{K}/\mathcal{N}$, i.e., $\mathcal{K}$ bits of information are transferred for each $\mathcal{N}$ physical bits transmitted. However, this rate is only achievable over a noiseless channel with $Cap(\mathcal{C}) = 1$. We may describe the rate of communication for a block code over a specific channel as $\mathcal{R}_{inf} = \mathcal{R} \times Cap(\mathcal{C})$. The subscript inf relates to the fact that the $Cap(\mathcal{C})$ is the *average* maximal transmission rate of the channel $\mathcal{C}$, thus $\mathcal{R}_{inf}$ is the theoretical transmission rate of the code over the channel for an infinite series of blocks.

The codewords of the block code do not necessarily have to contain the original sequence of the source bits. A block code that does contain the original sequence is called a systematic code. These codewords may be separated into the source bits and parity bits. A block code that does not have these properties is a non-systematic block code.

Linear codes may be written compactly in terms of matrices. Thus, the generation of the transmitted codewords $\mathbf{t}$ from the source blocks $\mathbf{z}$ is done using a generator matrix $\mathbf{G}$ such that

$$\mathbf{t} \stackrel{\text{def}}{=} \mathbf{zG}.$$

Once the codeword has been transmitted to the receiver, it is decoded using a parity-check matrix $\mathbf{H}$. The function of $\mathbf{H}$ is to compare the bits in the received block $\mathbf{r}$ in order to determine whether it is a valid codeword. The product of $\mathbf{H}$ and $\mathbf{r}$ thus produces a *syndrome vector* $\mathbf{s}$ as follows:

$$\mathbf{s} \stackrel{\text{def}}{=} \mathbf{rH}^T \tag{2.5.1}$$

where the exponent 'T' indicates the transposition of the vector/matrix and $\mathbf{s}$ indicates any possible incoherency in $\mathbf{r}$. This is known as syndrome decoding [2, p. 10].

2.5.2 The (7,4) Hamming code

Note that as we develop our understanding of Probabilistic Graphical Models (PGMs) through the remainder of this thesis, we will repeatedly use the (7,4) Hamming code as a basic example in order to illustrate the concepts at hand. Therefore, we start here by introducing the Hamming code from the viewpoint of decoding for linear codes. However, we will extend these concepts to the PGM domain as we progress through the thesis.

The (7,4) Hamming code is one of the most basic FEC-codes. Specifically, we will consider the systematic (7,4) Hamming code, although non-systematic versions are also available [16]. Each codeword has a length of $\mathcal{N} = 7$, with $\mathcal{K} = 4$ source bits and $\mathcal{L} = 3$ parity bits. This is the smallest form the Hamming code can take, with only $2^4 = 16$ possible codewords, and is the only form of the Hamming code we will consider in this thesis. Therefore, for the sake of brevity, we will from here onwards refer to the (7,4) Hamming code only as *the Hamming code*.

Traditionally the Hamming code is designed to be able to detect and correct a single bit error[5]. It does this by defining a fixed eXclusive-OR (XOR) relationship between the source bits and the parity bits. Thus, if the source bits are represented by $\mathbf{z} = z_0 z_1 z_2 z_3$ and the parity bits by $\mathbf{p} = p_0 p_1 p_2$, they will have the following relationships:

$$p_0 = z_0 \oplus z_1 \oplus z_2 \tag{2.5.2a}$$

$$p_1 = z_1 \oplus z_2 \oplus z_3 \tag{2.5.2b}$$

$$p_2 = z_0 \oplus z_2 \oplus z_3. \tag{2.5.2c}$$

where $x \oplus y$ is the XOR operation between x and y. For example, if we have the source bits $\mathbf{z} = 1100$, the parity bits are

$$p_0 = 1 \oplus 1 \oplus 0 = 0$$
$$p_1 = 1 \oplus 0 \oplus 0 = 1$$
$$p_2 = 1 \oplus 0 \oplus 0 = 1$$

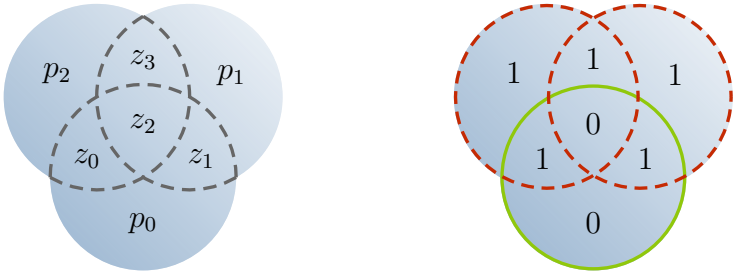
[5]However, when using PGMs this limit does not apply, as is shown in Chapter 4

and the Hamming codeword would be $\mathbf{t} = \mathbf{z}\|\mathbf{p} = 1100011$. We may use matrix notation to express this where the generation matrix of the Hamming code is defined as:

$$\mathbf{G} = \begin{array}{c} \\ z_0 \\ z_1 \\ z_2 \\ z_3 \end{array} \begin{array}{c} \begin{array}{ccccccc} z_0 & z_1 & z_2 & z_3 & p_0 & p_1 & p_2 \end{array} \\ \left[\begin{array}{ccccccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{array}\right] \end{array}.$$

For systematic block codes we may define $\mathbf{G}$ as the combination of two sub-matrices: an identity matrix $\mathbf{I}_{\mathcal{K}}$ of size $\mathcal{K} \times \mathcal{K}$ and a matrix $\mathbf{P}$ of which the columns coincide with the equations in (2.5.2) such that

$$\mathbf{G} = \left[\begin{array}{cc} \mathbf{I}_4 & \mathbf{P} \end{array}\right].$$



(a) Diagram of the parity bit dependencies

(b) An example with a bit flip at z_3

Figure 2.13: Venn diagrams of the Hamming code. (a) Each circle contains a subset of the 7 variables which relate to one another according to (2.5.2). (b) shows an example where the bit sequence $\mathbf{r} = 1101011$ was received. The top 2 circles detect an error, thus bit z_3 must be flipped in order to repair the original codeword $\mathbf{t} = \mathbf{z}\|\mathbf{p} = 1100011$.

The generation process can best be illustrated using Venn diagrams as in Fig. 2.13(a). Here each circle contains 3 source bits and one parity bit according to equations (2.5.2). From these equations we see that if a correct codeword is received, the XOR of all 4 bits in each circle should equal 0. With this in mind we define the following syndrome equations:

$$s_0 = p_0 \oplus z_0 \oplus z_1 \oplus z_2 \tag{2.5.3a}$$

$$s_1 = p_1 \oplus z_1 \oplus z_2 \oplus z_3 \tag{2.5.3b}$$

$$s_2 = p_2 \oplus z_0 \oplus z_2 \oplus z_3 \tag{2.5.3c}$$

where s_0 to s_2 are called the syndromes of the Hamming code.

The decoding of a codeword at the receiver is done by inspecting these relationships to ensure they are valid. Thus for a valid codeword, the syndromes s_0 to s_2 will equal 0. Let us consider Fig. 2.13(b) as an example where the bit sequence $\mathbf{r} = \mathbf{z}\|\mathbf{p} = 1101011$ was received. The syndrome tokens will then be

$$\begin{aligned} s_0 &= 0 \oplus 1 \oplus 1 \oplus 0 = 0 \\ s_1 &= 1 \oplus 1 \oplus 0 \oplus 1 = 1 \\ s_2 &= 1 \oplus 1 \oplus 0 \oplus 1 = 1. \end{aligned}$$

Clearly there is an error as both s_1 and s_2 equal 1. It turns out that for each one of the 7 possible bit flips, the syndrome vector $\mathbf{s} = \begin{bmatrix} s_0 & s_1 & s_2 \end{bmatrix}$ is unique [2]. This allows us to identify the outlier and correct it. Table 2.2 shows the 7 possible bit flips and their respective syndrome vectors.

Table 2.2: The 7 possible bit flips of the Hamming code and their respective syndrome vectors

Bit flip	z_0	z_1	z_2	z_3	p_0	p_1	p_2
$\mathbf{s}$	101	110	111	011	100	010	001

Again, we may simplify the process by using matrix algebra. To do this we use the $\mathcal{L} \times \mathcal{N}$ parity-check matrix $\mathbf{H}$, where each row corresponds to a unique syndrome and each column to a unique bit. We then place a 1 in each row of the matrix for each bit the respective syndrome expresses, according to equations (2.5.3), and fill the rest with zeros such that

$$\mathbf{H} = \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \end{array} \begin{array}{c} \begin{array}{ccccccc} z_0 & z_1 & z_2 & z_3 & p_0 & p_1 & p_2 \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \end{array}.$$

It may be noted that the columns of the resulting matrix correspond to the syndrome vector of the respective bit as seen when compared to Table 2.2. We can now express the syndrome vector as $\mathbf{s} = \mathbf{r}\mathbf{H}^T$, for which all valid codewords produce a zero syndrome vector.

With this topography we can only detect 1 bit error, as a second error will lead the syndrome check to deduce the wrong answer. To illustrate this, let us examine another example where we transmit the codeword $\mathbf{t} = 1100011$

and receive the bit sequence $\mathbf{r} = 1001011$. The syndrome check thus needs to detect that bits z_1 and z_3 were flipped. However, the resulting syndrome vector is

$$\mathbf{s} = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}.$$

From Table 2.2 this vector indicates incorrectly that bit z_0 should be flipped to produce the codeword $\mathbf{r} = 0001011$. The number of bits an FEC-code can correct is known as its *hamming distance*.

2.5.3 Low-Density Parity-Check codes

LDPC codes were first proposed by Robert Gallager in 1962 [29]. However, at the time computer technology was incapable of performing Gallager's highly complex decoding algorithm for the LDPC code cost effectively. Thus, it was neglected for approximately 35 years [30]. In the 1990s it was rediscovered by McKay and Neal following the revolution in coding theory that was resulted from the introduction of turbo codes.

Like the Hamming code, the LDPC code is a type of linear block code that encodes a set of source bits $\mathbf{z} = \begin{bmatrix} z_0 & z_1 & \dots & z_{\mathcal{K}-1} \end{bmatrix}$ into a codeword $\mathbf{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_{\mathcal{N}-1} \end{bmatrix}$ where $\mathcal{N} > \mathcal{K}$. The construction of an LDPC code is defined by a set of *parity-check* equations. These equations are the XOR summation of a subset of the codeword bits such that

$$\bigoplus_{t_j \in \mathbf{t}_i} t_j = 0$$

for $i = 0, 1, \dots, \mathcal{L} - 1$, where $\mathcal{L}$ is typically smaller than $\mathcal{N}$ and $\mathbf{t}_i \subset \mathbf{t}$. These equations are the same as the syndrome equations that were used for the Hamming code. In fact, the Hamming code may be considered as a special case LDPC code, where the parity-check equations that define the codeword

are

$$t_0 \oplus t_1 \oplus t_2 \oplus t_4 = 0$$
$$t_1 \oplus t_2 \oplus t_3 \oplus t_5 = 0$$
$$t_0 \oplus t_2 \oplus t_3 \oplus t_6 = 0.$$

Similar to the syndrome tokens of the Hamming code, these equations can be defined using the parity-check matrix $\mathbf{H}$ such that

$$\mathbf{t}\mathbf{H}^T = \mathbf{0}.$$

To generate the codeword from the source bits, we require the generation matrix $\mathbf{G}$ such that $\mathbf{t} = \mathbf{z}\mathbf{G}$. A property of linear block codes is that

$$\mathbf{G}\mathbf{H}^T = 0 \text{ [30]}.$$

Thus, having defined a given LDPC code by its parity-check matrix $\mathbf{H}$, this equation may be used to obtain the generation matrix.

When designing the parity-check matrix $\mathbf{H}$, the number of 1s in each column of $\mathbf{H}$ is often defined as a fixed value ω_c, referred to as the column weight. The same applies to the rows and row weight ω_r. For this to be possible the size of $\mathbf{H}$ must be $(c\,\omega_c) \times (c\,\omega_r)$, where c is an integer. A code with these constraints is known as a (ω_c, ω_r)-regular LDPC code [31].

Provided that all the rows of $\mathbf{H}$ are linearly independent, the *design rate* of a (ω_c, ω_r)-regular code is $\mathcal{R} = 1 - \omega_c/\omega_r$. However, this is often not the case and thus we may expect the actual code rate to be slightly higher [16].

Assuming that the codeword $\mathbf{r} = \begin{bmatrix} r_0 & r_1 & \dots & r_{\mathcal{N}-1} \end{bmatrix}$ was received, the receiver may check if it is a valid codeword by calculating the expression in (2.5.1):

$$\mathbf{s} = \mathbf{r}\mathbf{H}^T$$

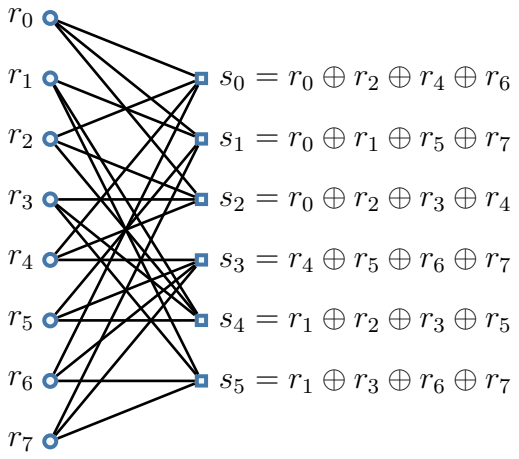
where $\mathbf{s}$ is the syndrome vector that is a zero-vector if $\mathbf{r}$ is a valid codeword.

An example of a (3,4)-regular LDPC code is shown in Fig. 2.14. The graph in Fig. 2.14(b) is known as a Tanner graph, which is a representation of the $\mathbf{H}$ matrix in Fig. 2.14(a). Each circle in the graph represents a bit in the codeword, i.e., a column in $\mathbf{H}$. The squares represent the bits of the syndrome vector. The circle i is connected to square j if $\mathbf{H}_{i,j} = 1$.

If $\mathbf{s} \neq \mathbf{0}$ the receiver can attempt to recover to original transmitted codeword $\mathbf{t}$ by applying decoding algorithms such as Belief Propagation (BP) to

$$\mathbf{H}= \begin{array}{c} \\ s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{array} \begin{array}{c} \begin{array}{cccccccc} r_0 & r_1 & r_2 & r_3 & r_4 & r_5 & r_6 & r_7 \end{array} \\ \left[\begin{array}{cccccccc} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right] \end{array}$$

(a) Parity-check matrix



(b) Tanner graph

Figure 2.14: The Parity-check matrix $\mathbf{H}$ and the equivalent Tanner graph of a (3,4)-regular LDPC code with design rate $\frac{1}{4}$. (b) Each circle represents a bit in the codeword $\mathbf{r} = \begin{bmatrix} r_0 & r_1 & \dots & r_7 \end{bmatrix}$ and a column in $\mathbf{H}$. The squares represent the check bits $\mathbf{s} = \begin{bmatrix} s_0 & s_1 & \dots & s_5 \end{bmatrix}$. The circle i is connected to square j if $\mathbf{H}_{i,j} = 1$. This graph is used for decoding the LDPC code, as we will see in chapter 4.

the graph, where the graph is decoded locally at each step. The decoding is successful when $\mathbf{s} = \mathbf{0}$. Exactly how this is done is fully explained in Chapter 4.

It is due to the BP decoding approach that $\mathbf{H}$ needs to be sparse, as the decoding complexity is directly proportional to ω_c. However, the minimum Hamming distance of the LDPC code is linearly and *inversely* proportional to ω_c, as long as $\omega_c \geq 3$ [16]. Thus, a balance between these 2 factors is required to produce a good LDPC code.

LDPC may also be irregular, as introduced by Luby [32], which implies that the column and row weights may differ throughout $\mathbf{H}$. It turns out that irregular codes preform better over BECs; however, for BAWGNCs, regular code are superior. Furthermore, constructing $\mathbf{H}$ to be as random as possible while remaining within the constraints generally improves performance.

2.6 Digital fountain codes

With low-complexity approximating decoding techniques, block codes perform close to channel capacity. However, these codes have a few design limitations that render them unsuitable for broadcasting [8]. Digital fountain codes are a class of codes designed to overcome these limitations while maintaining linearity and low-complexity decoding.

One of the design limitations fountain codes address is due to the fixed code rates of the block codes. A block code needs to be designed with a specific rate $\mathcal{R}$ *prior* to transmission, thus the parameters of the channel have to be known beforehand. This leads to inefficiencies when communicating over a channel with unknown parameters, or one of which the capacity is not constant due to fluctuating noise levels, both typical properties of broadcasting systems. These inefficiencies are due to the fact that, by definition, the rate of communication needs to be below the channel capacity to achieve arbitrary small error probability, and the only fixed code rate that guarantees this over all possible channels is $\mathcal{R} = 0$ [8].

Should one forgo the design for arbitrary channels and design for a channel with varying capacity, one may compensate for the lack of knowledge by designing the code for the worst case scenario. Although this approach assures reliable communication, it also further limits the code's performance below the channel capacity than would otherwise be possible.

Another limitation of block codes is that they require that all packages be received. This is due to the independence of all the transmitted packets, meaning each packet contains no information pertaining to any other packet. Subsequently, should any packet be dropped it would leave the receiver with incomplete data. This becomes a problem for broadcasting when, in an extreme case, each packet is dropped by at least 1 receiver. In this case the entire data file needs to be transmitted twice, effectively halving the communication rate.

These problems were overcome by introduction of the *digital fountain approach* in [33, 34], which is designed for communication over a BEC with unknown erasure probability. What makes fountain codes appropriate for broadcasting is that they are naturally *rateless*, i.e., for a given finite set of source packets a fountain code can theoretically generate an infinite sequence of coded output packets. This is achieved while retaining low complexity for

both the encoding and decoding.

To explain the digital fountain approach, the analogy of a water fountain is often used [8, 33], [2, p. 590]. The metaphor represents the receiver as a person wishing to fill a bucket with water by standing beneath a fountain, which represents the transmitter. In this instance it does not matter which water droplets are caught in the bucket, as long as enough droplets are caught. Similarly, when using digital fountain codes it does not matter which packets are received. As long as enough packets were obtained, the source data may be recovered.

2.6.1 The general formulation of fountain codes

Similar to the linear block codes, the encoding of the transmitted symbols of a binary fountain code is defined by

$$\mathbf{t} = \mathbf{z}\mathbf{G} \tag{2.6.1}$$

with the source symbol set $\mathbf{z} = \begin{bmatrix} z_0 & z_1 & \dots & z_{\mathcal{K}-1} \end{bmatrix}$ and transmitted symbol set $\mathbf{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_{\mathcal{M}-1} \end{bmatrix}$, where $\mathcal{M} \to \infty$. Moreover, each transmitted symbol (t_i) is an XOR summation of a random subset of the source symbols $\mathbf{z}$ such that

$$t_i = \bigoplus_{z_j \in \mathbf{z}_i} z_j$$

for $i = 0, 1, \dots, \mathcal{M}$ where $\mathbf{z}_i \subset \mathbf{z}$. The source symbols of which each transmitted symbol is equal to its check-sum are indicated by the 1s in each column of the generator matrix $\mathbf{G}$. The number of 1s in each column is known as the respective transmitted symbol's *degree* $\mathcal{D}_o$, a.k.a. its weight.

For an *infinite* sequence of transmission symbols, the information of the source data is distributed *evenly* across these symbols. It is this distribution of the information that allows the receiver to receive any combination of the transmitted symbols in any order. That is, even should some of the transmitted symbols be lost, all of the source data will still be available to the receiver via the symbols that were transmitted successfully. Subsequently, fountain codes are robust against erasures and are rateless.

However, the random element of the transmission symbols' construction presents a problem for the decoder. With block codes the decoder may know the structure of the generator matrix prior to transmission, but in practice

fountain codes are generated on-the-fly. There are many solutions presented for this in [10, 35]. For example, each transmitted symbol may be augmented with the list of source symbols that was used to calculate that symbol. It is also possible for the decoder to compute this list implicitly based on properties such as the unit time at which the data was received.

To reduce the overhead caused by these bookkeeping operations, fountain codes may be encoded using parallel concatenation [8]. This is done by dividing the entire set of source bits

$$\mathbf{b} = \begin{bmatrix} b_0 & b_1 & \dots & b_{\mathcal{S}-1} \end{bmatrix}$$

into subsets. Each of these subsets is subsequently handled as a unique symbol such that

$$z_i = \begin{bmatrix} b_{il} & b_{il+1} & \dots & b_{il+l-1} \end{bmatrix}$$

for $i = 0, 1, \dots, \mathcal{K}-1$ where $\mathcal{K} = \mathcal{S}/l$. Moreover, the vector of source symbols is given as

$$\mathbf{z} = \begin{bmatrix} z_0 & z_1 & \dots & z_{\mathcal{K}-1} \end{bmatrix} = \begin{bmatrix} z_0 & z_1 & \cdots & z_{\mathcal{K}-1} \\ b_0 & b_l & \cdots & b_{(\mathcal{K}-1)l} \\ b_1 & b_{l+1} & \cdots & b_{(\mathcal{K}-1)l+1} \\ b_2 & b_{l+2} & \cdots & b_{(\mathcal{K}-1)l+2} \\ \vdots & \vdots & \ddots & \vdots \\ b_{l-1} & b_{2l-1} & \cdots & b_{\mathcal{S}-1} \end{bmatrix}.$$

Similarly we have

$$\mathbf{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_{\mathcal{M}-1} \end{bmatrix} = \begin{bmatrix} t_0 & t_1 & \cdots & t_{\mathcal{M}-1} \\ c_0 & c_l & \cdots & c_{(\mathcal{M}-1)l} \\ c_1 & c_{l+1} & \cdots & c_{(\mathcal{M}-1)l+1} \\ c_2 & c_{l+2} & \cdots & c_{(\mathcal{M}-1)l+2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{l-1} & c_{2l-1} & \cdots & c_{(\mathcal{M}-1)l+l} \end{bmatrix}.$$

Consequently, the fountain code may be designed to encode a data block of only size $\mathcal{K}$ instead of $\mathcal{S}$.

The symbols compiled from the source data are known as the *source* symbols, where the transmitted symbols are known as the *output* symbols. The size of these symbols holds no bearing on the theory [10], thus for simplification we will assume $l = 1$ for the remainder of the thesis.

Furthermore, fountain codes are said to be *universal* codes [8, 10, 18], implying that they are able to perform close to capacity for any BEC. The only discrepancy between the code rate and the BEC capacity is the difference between the number of source symbols $\mathcal{K}$ and output symbols $\mathcal{M}$. Part of the design of a fountain code is to minimise this overhead such that $\mathcal{M} \to \mathcal{K}$.

2.6.2 The random linear fountain code

One of the most theoretically fundamental implementations of the fountain code is the *random linear fountain code* [8, 18]. The generator matrix $\mathbf{G}$ in (2.6.1) of the random linear fountain code is constructed with its binary values defined uniformly at random. Furthermore, the output symbols are generated on-the-fly, as opposed to linear block codes which are constructed prior to transmission. Each newly constructed symbol adds a column to $\mathbf{G}$.

The random linear fountain code employs Gaussian Elimination (GE) to recover the source symbols. The recovery will thus only be possible if the rank of $\mathbf{G}$ is $\mathcal{K}$. This implies that $\mathcal{M} \geq \mathcal{K}$ is required for successful recovery of the source data. Since we wish to achieve a code rate close to capacity, we aim to keep $\mathcal{M}$ as small as possible while retaining a good decoding probability. As shown in [18], the probability for a random $\mathcal{K} \times \mathcal{K}$ matrix to be fully ranked is

$$\prod_{i=1}^{\mathcal{K}} \left(1 - 2^{-i}\right) \approx 0.2888$$

for any $\mathcal{K} \geq 10$. If $\mathcal{M} = \mathcal{K}$, this is equal to the probability of successful decoding, which is a tremendously poor performance.

However, the postulation is made by Shokrollahi *et al* in [20] that, for a small overhead o such that $\mathcal{M} = \mathcal{K} + o$, the failure probability δ has the upper bound of 2^{-o}. Remarkably, this upper bound is found to be independent of the size of $\mathcal{K}$. Thus, for large values of $\mathcal{K}$, an overhead of relatively trivial size is required for a high probability of success.

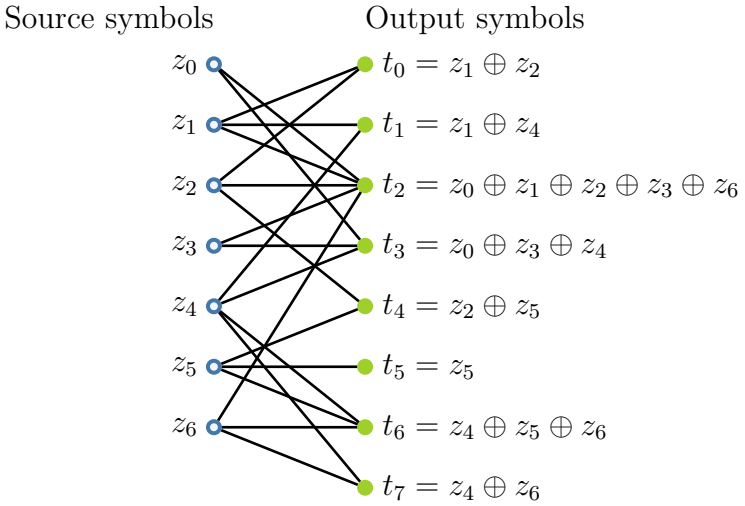
Unfortunately, the random linear fountain code only applies to BECs, since GE is incapable of correcting errors within the symbols. Furthermore, the decoding complexity is $O(\mathcal{K}^3)$, as proven in [8], due to the use of GE. This decoding complexity proves to be too high in practice, making the random linear fountain code an insufficient solution.

2.6.3 The Luby Transform code

The first practical realisation of a fountain code is the Luby Transform (LT) code as introduced by M. Luby in [10]. The LT code is an improvement on the random linear fountain code, where the complexity is lowered by creating a sparse matrix $\mathbf{G}$ as well as replacing the GE with an approximate decoding algorithm. This decoding algorithm is one that is appropriate for decoding sparse linear systems and is able to find a local optimum solution.

$$\mathbf{G} = \begin{array}{c} \\ z_0 \\ z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \\ z_6 \end{array} \begin{array}{c} \begin{array}{cccccccc} t_0 & t_1 & t_2 & t_3 & t_4 & t_5 & t_6 & t_7 \end{array} \\ \begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{array}$$

(a) Generator matrix



(b) Tanner graph

Figure 2.15: An example of an LT code with $\mathcal{K} = 7$ source symbols and $\mathcal{M} = 8$ output symbols. (a) shows the code's generator matrix $\mathbf{G}$ and (b) shows the equivalent graph, where the source symbols are depicted on the left side and the output symbols on the right side.

To make $\mathbf{G}$ more sparse while maintaining a random construct, each output symbol's degree $\mathcal{D}_o$ is sampled at random from a PD called the *degree distri-*

bution, the design of which we discuss in Section 2.6.3.2. The source symbols on which each output symbol is dependent are chosen uniformly at random.

Figure 2.15 shows such an example with $\mathcal{K} = 7$ and $\mathcal{M} = 8$. The graph in Fig. 2.15(b) is similar to the Tanner graph in Fig. 2.14(b). However, the edges here are connected between source symbols and output symbols, indicating their respective dependencies.

Initially, all the information that is available to the receiver is contained within the output symbols, due to the observation of their outcomes as they were received. In the following sub-section we will illustrate how this is used to decode the source symbol when transmitted across a BEC.

2.6.3.1 The LT-process

The LT process was defined in [10] in order to describe the design of the degree distribution for the LT codes. However, the LT process is effectively the decoding algorithm of an LT code. We closely follow MacKay's explanation of the LT process in [18] in this section.

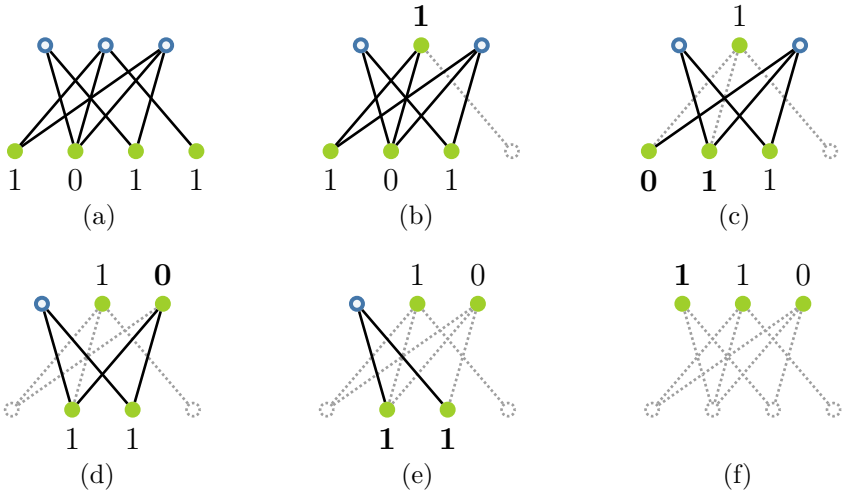


Figure 2.16: The LT process as the decoding algorithm, where the nodes of which the receiver knows its value are depicted as a solid green circle. (a) shows the initial graph where 4 output symbols that encode 3 input symbols were received. In (b) the first input symbol is decoded and the respective output symbol removed. Subsequently, in (c) the dependencies on the decoded input symbol are removed. The process continues until all degree 1 output symbols are exhausted (d)–(f).

Consider the graph in Fig. 2.16(a), where 4 output symbols that encode

3 source symbols were received. The initial step collects all output symbols of degree $\mathcal{D}_o = 1$ and uses them to decode their unique dependant, i.e., their value is assigned to the respective source symbol. These output symbols are then removed from the graph as seen in Fig. 2.16(b).

Subsequently, the decoded source symbols are removed from their output dependants by means of an XOR operation. For example, if the source symbol has a value of 1 and the dependant output symbol 0, the dependency (edge) is removed and the output symbol's value becomes $1 \oplus 0 = 1$. Two such cases that are depicted in bold can be found in Fig. 2.16(c).

Thus, the success of the LT process depends on each iteration producing at least one new output symbol of degree $\mathcal{D}_o = 1$. The decoding process continues until all such output symbols are exhausted. The LT process fails if there is at least one source symbol that has not been decoded after its termination.

This algorithm is specifically designed for the behaviour of a BEC, since all received output symbols are presumed to be correct.

2.6.3.2 Degree distributions for the LT-code

The question arises how to obtain a graph structure that ensures the LT process would be successful, since the output symbols' encoding is randomised. To accomplish this, their encoding may be defined indirectly by choosing the appropriate distributions that define how the randomised encoding is done.

There are two parts of the encoding that are independently randomised for each output symbol: the degree $\mathcal{D}_o$ of each output symbol, and which subset $\mathbf{z}_i \subset \mathbf{z}$ of the source symbols set $\mathbf{z}$ they encode. The latter is chosen using a uniform distribution to ensure an even distribution of the information across all output symbols. However, it is the degree distribution that has an influence on the success of the LT process.

For the LT process to be optimal, no redundancy may exist in the structure of the graph, while the LT process is allowed to successfully decode all the source symbols. This implies that one, and only one, output symbol must be of degree 1 at every step of the LT process. For the theoretical LT code with endless output symbols, this is achievable by using the *ideal soliton distribution*

[10, 18]: a probability mass function defined as

$$\Omega_{ideal}(\mathcal{D}_o) \stackrel{\text{def}}{=} \begin{cases} \dfrac{1}{\mathcal{K}} & \text{for } \mathcal{D}_o = 1 \\ \dfrac{1}{\mathcal{D}_o(\mathcal{D}_o - 1)} & \text{for } \mathcal{D}_o = 2, 3, \ldots, \mathcal{K} \end{cases}$$

where $\mathcal{D}_o$ is the degree of the output symbols and $\mathcal{K}$ is the total number of source symbols. As state by M. Luby [10], the development of this distribution was inspired by a soliton wave [36] and is designed to have an average output degree of $\mathcal{D}_{ave} = \ln(\mathcal{K})$, which is needed to maintain a high probability that all input symbols are encoded at least once. This implies that the encoding and decoding complexities scale with $\mathcal{K}\ln(\mathcal{K})$. The development of the ideal soliton distribution may be found in [8, 20]. A depiction of the distribution may be found in Fig. 2.17 for $\mathcal{K} = 1000$.

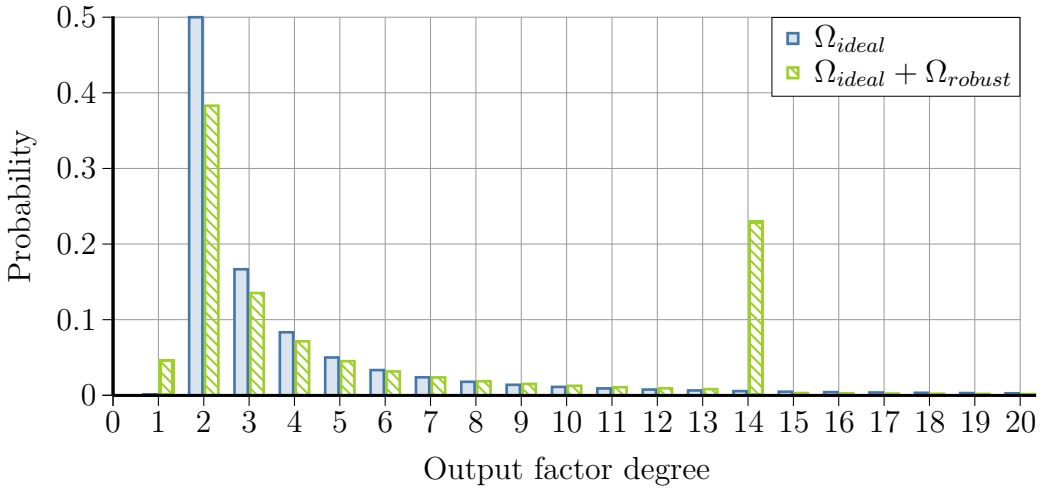


Figure 2.17: The soliton distributions that form the optimal degree distribution for LT codes over BECs. The ideal soliton is designed for a code with infinite output symbols. Because this is not possible in practice, the robust soliton is combined with the ideal to compensate for a finite output. Due to its small value, the $\mathcal{D}_o = 1$ component of the ideal soliton is present but not visible in this figure.

In practice, this distribution is very fragile since there will always only be a finite number of output symbols. This makes it very likely that fluctuations from the ideal behaviour may cause the LT process to fail. So much so that, with a finite number of output symbols, even a few nodes of degree 1 may not be enough to guarantee successful decoding. As a result, the ideal soliton distribution performs poorly in practice.

In [10] some adjustments were made to improve the distribution's performance in practice by increasing the redundancy slightly. The result is called the *Robust soliton distribution* and is defined as follows:

$$\Omega_{robust}(\mathcal{D}_o) \stackrel{\text{def}}{=} \begin{cases} \dfrac{S}{\mathcal{D}_o \mathcal{K}} & \text{for } \mathcal{D}_o = 1, 2, \ldots, (\mathcal{K}/S - 1) \\ \dfrac{S}{\mathcal{K}} \ln\left(\dfrac{S}{\delta}\right) & \text{for } \mathcal{D}_o = \mathcal{K}/S \\ 0 & \text{otherwise} \end{cases}$$

where $S \equiv c \ln(\frac{\mathcal{K}}{\delta})\sqrt{\mathcal{K}}$. δ is the specified failure probability of decoding and c is some real constant such that $c \in (1, 0)$. c is a free parameter typically used to tune the distribution for specific cases.

These two distributions are then combined by addition $(\Omega_{ideal} + \Omega_{robust})$ and normalised to give us a new distribution shown in Fig. 2.17. It is shown in [18] that, with the new distribution, a probability for successful decoding of $1 - \delta$ may be possible with as little as $\mathcal{M} \approx \mathcal{K} + 2\ln(S/\delta)S$ received output symbols.

2.6.4 Raptor codes

As previously discussed, for reliable decoding of the LT code the decoding complexity increases *super-linearly* with $\ln(\mathcal{K})$ for each added output symbol. In this thesis we will focus on an extension of the LT code called the Raptor (*rap*id-*tor*nado) code. The Raptor code improves on the LT code by *weakening* it in such a manner that its complexity only increases *linearly* with $\mathcal{K}$ [18].

That is, the Raptor code uses a distribution that induces an average degree $\mathcal{D}_{ave}$ for the output symbols that is intentionally kept small in order to ensure a low complexity, thus $\mathcal{D}_{ave} < \mathcal{K}\ln(\mathcal{K})$. However, we know that it is required that $\mathcal{D}_{ave} \geq \mathcal{K}\ln(\mathcal{K})$ in order to assure that all source symbols are encoded at least once. As a result, it is likely that the Raptor code will have some fraction of the source symbols that are not encoded.

To deal with the fraction of source symbols missed by the weakened LT code, the Raptor codes encode the source symbols with a separate sparse-graph code. This encoding is defined as

$$\mathbf{y} = \mathbf{z}\mathbf{G}_y,$$

where $\mathbf{z} = \begin{bmatrix} z_0 & z_1 & \dots & z_{\mathcal{K}-1} \end{bmatrix}$ and $\mathbf{y} = \begin{bmatrix} y_0 & y_1 & \dots & y_{\mathcal{N}-1} \end{bmatrix}$. We will refer to this sparse-graph code as the *pre-code*, which may be such codes as the LDPC code for example. Only after this first encoding are these new symbols encoded with the LT code such that

$$\mathbf{t} = \mathbf{y}\mathbf{G}_t,$$

where $\mathbf{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_{\mathcal{M}-1} \end{bmatrix}$. Subsequently, output symbols $\mathbf{y}$ of the pre-code will be referred to as the *input symbols*, since they form the input of the LT-section of the Raptor code.

A depiction of such a Raptor code may be found in Fig. 2.18, where $\mathcal{K} = 4$, $\mathcal{M} = 8$ and the pre-code is the (7,4) Hamming code.

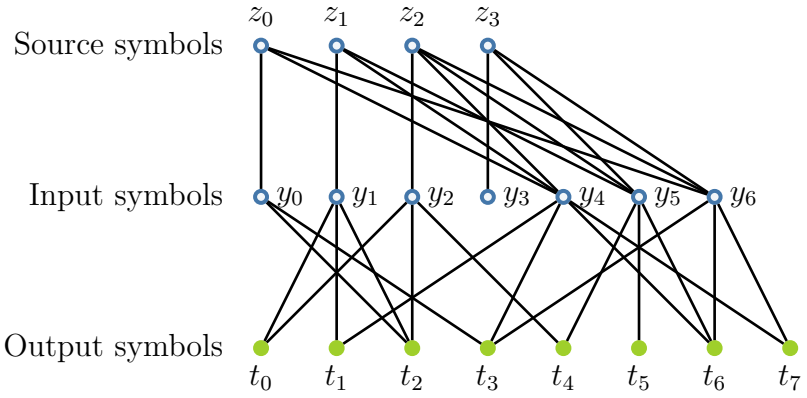


Figure 2.18: An example of a Raptor code with $\mathcal{K} = 4$ and $\mathcal{M} = 8$. The source symbols $\mathbf{z}$ are encoded using the Hamming code as a pre-code. The resulting input symbols $\mathbf{y}$ are then encoded using a weakened LT code with an average output degree of 2.25 to obtain the outputs $\mathbf{t}$. Note that y_3 is missed by the LT code, yet remains recoverable through the pre-code.

If the degree distribution of the weakened LT code is designed correctly, only a small fraction of the source symbols will not be encoded. As is proven in [10, 18], this fraction is close to $e^{-\mathcal{D}_{ave}}$. The pre-code must therefore be chosen so that it can decode this small fraction of input symbols, typically a right-regular LDPC code.

The design of the degree distribution for the Raptor code is done by asymptotic linear programming known as density evolution [9]. This is not a topic covered in this thesis, however the distribution designed for the standardised R10 Raptor code in [8] is as follows:

Table 2.3: The degree distribution of the R10 Raptor code, with an average degree of 4.63. This is less than the required $\ln(\mathcal{K}) = 6.91$ for the LT code, as expected.

$\Omega(\mathcal{D}_o)$	0.0098	0.4590	0.2110	0.1134	0.1113	0.0799	0.0156
$\mathcal{D}_o$	1	2	3	4	10	11	40

2.6.5 Universality of Raptor codes

Previously we stated that fountain codes are universal codes, implying that they are able to perform close to capacity for any BER. However, this statement is made under the assumption that the code is applied over a BEC. Considering that we wish to apply the Raptor code to the BSC and BAWGNC, it needs to establish whether the above statement holds for these channels as well.

In the work of Etesami and Shokrollahi in [17] the value of the $\mathcal{D}_o = 2$ component of the degree distributions for the BEC, BSC and BAWGNC that achieve channel capacity for all possible BERs are determined. The plots in Fig. 2.19 depict their results. We do not need to analyse the equations that define these plots to make the necessary observation; however, a brief overview of their definition and deviration may be found in Appendix B.

Fig. 2.19 depicts the channel models degree distributions $\Omega_{\mathcal{C}}$ for the channel models $\mathcal{C} = \{\text{BEC}, \text{BSC}, \text{BAWGNC}\}$. We see that $\Omega_{\text{BEC}}(2)$ is a constant for any erasure probability, supporting the claim that Raptor codes are universal for BECs. Unfortunately, we see that this is not the case for BSCs and BAWGNCs, thus concluding that Raptor codes are not universal for these channels.

These observation are important, since they imply that we need a different degree distribution for different BERs. This inhibits our ability to communicate at channel capacity for BSCs and BAWGNCs when the channel parameters are unknown. However, it can be shown that

$$\frac{1}{\ln(16)} \leq \Omega_{\mathcal{C}}(2) \leq \frac{1}{2},$$

which is a reasonably small variation.

Accordingly, Etesami and Shokrollahi conjecture that degree distributions optimised for BECs perform reasonably well, although not optimally, over other BIMSCs. In light of this, the design of degree distributions for the BAWGNC and BSC and how to deal with unknown BER are not covered in this thesis. Subsequently, we will continue to use the degree distributions

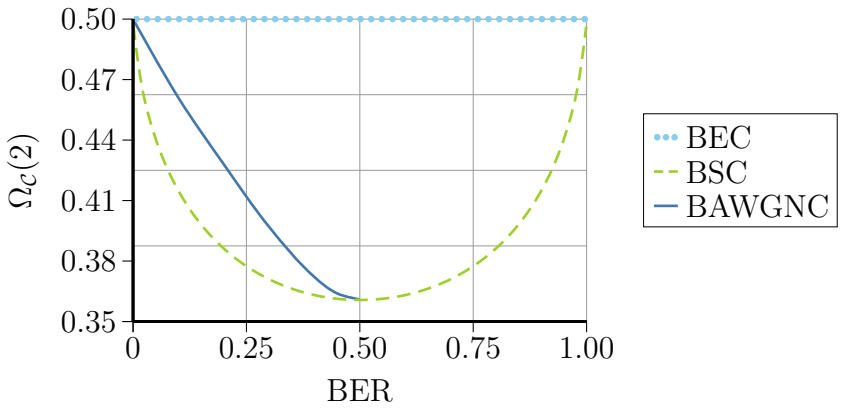


Figure 2.19: The bounds on $\Omega_{\mathcal{C}}(2)$ for the BEC, BSC, and BAWGNC. We see that this is a constant for the BEC, an indication that Raptor codes are universal for erasure channels. However, for the BSC and BAWGNC, this is not the case and thus we conclude that the Raptor code is not universal for these channels.

optimised for BECs for our simulations on all channels while acknowledging the possibility for improvement.

2.7 Summary

This chapter covered the necessary information theory needed for the work to follow. Each section's theory may relate back to one of the six stages depicted in Fig. 2.1. The scope of the thesis is limited to binary data and BIMSCs.

The three BIMSCs that were covered in Section 2.2 are the BEC, BSC, and BAWGNC. The BEC emulates the loss of packets with an erasure probability ϵ, whereas the BSC and BAWGNC emulate the introduction of errors in packets with bit flip probability ρ and noise variance ${\sigma_n}^2$, respectively.

In Section 2.3, we found that it is possible to measure the amount of information we receive by means of entropy. Through the concept of mutual information it was shown how to calculate how much information we gain about the input, when the output of a channel is observed. Using this, the capacities of the three channel models were defined in Section 2.4. These capacities define the minimum redundancy required to successfully transfer the source data to the receiver as were illustrated in Figs. 2.11 and 2.12.

FEC-codes, as covered in Section 2.5, are an efficient way to add the necessary redundancy to the source data. These codes include linear block codes such as the Hamming and LDPC codes, as well as fountain codes such as

the LT- and Raptor codes. Linear block codes spread the information of $\mathcal{K}$ bits across $\mathcal{N}$ bits to improve the resilience of the data against noise. In contrast, fountain codes spread the source information across a potentially endless stream of bits, making the data robust against packet drops and burst errors. Error detection was also mentioned as a possible way to add the necessary redundancy to the source data.

The fixed rate of the linear block codes proves problematic when designing a code of a channel with unknown parameters. In Section 2.6, we found that fountain codes are universal for BECs, i.e., they are able to perform close to channel capacity for all ϵ. Unfortunately, fountain codes are not universal for BSCs and BAWGNCs. However, they perform close to channel capacity for these channels regardless and are thus suitable candidates for broadcasting.

An overview of the (7,4) Hamming code was given as a basic example to illustrate the concepts in the following chapter. In Chapter 3 we will cover the theory of PGMs and the different ways in which they may depict FEC codes. Subsequently, we will see how these PGMs may be used to decode the FEC codes by means of message-passing algorithms.

Chapter 3

Probabilistic Graphical Models

3.1 Introduction

The graphs presented in the previous chapter perform well when implemented alongside a deterministic decoding algorithm, e.g., the Luby Transform (LT)-process. However, for channels such as the Binary Symmetric Channel (BSC) or Binary Additive White Gaussian Noise Channel (BAWGNC), confidence in the received data is no longer absolute and thus a deterministic algorithm is no longer sufficient or optimal. Instead we need to propagate Probability Distributions (PDs) across the graphs via some probabilistic decoding algorithm.

When solving a probabilistic problem, the goal is to determine the posterior PD using the available information and ultimately derive some conclusion from it. However, for most practical applications, calculating this posterior PD directly is too computationally expensive. In this section we will investigate different Probabilistic Graphical Models (PGMs): a very generalised, model-based system that simplifies this task by representing it graphically, and by expressing the problem through factorisation.

Moreover, the depiction of the joint PD $P(\mathbf{x})$ by a PGM is the expression of the PD as a product of $\mathcal{W}$ factors [2, p. 334], such that

$$P(\mathbf{x}) \propto \prod_{w=0}^{\mathcal{W}-1} f_w(\mathbf{x}_w).$$

These factors may be a conditional or joint PD of a subset of variables in the scope of $P(\mathbf{x})$, that is $\mathbf{x}_w \subset \mathbf{x}$. The purpose of these factors is to fully describe the dependencies pertaining to the variables in a more efficient manner. Thus,

if 2 variables in $\mathbf{x}$ are independent, it is possible to define the factors in such a way that they are mutually exclusive in all of the subsets $\mathbf{x}_w$.

Note that the joint PD is not equal, but only proportional to the product of these factors. This is because $P(\mathbf{x})$ must be normalised, which is not a constraint of the factors, i.e., $\sum_{\mathbf{x}_w} f_w(\mathbf{x}_w) \neq 1$. This may be compensated for by normalising over the factors such that

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{w=0}^{\mathcal{W}-1} f_w(\mathbf{x}_w) \tag{3.1.1}$$

where

$$Z = \sum_{\mathbf{x}} \prod_{w=0}^{\mathcal{W}-1} f_w(\mathbf{x}_w).$$

Using the relationship of the variables to define these factors, we reduce the computational load necessary to compute the posterior PD while retaining generality. More specifically, our purpose for PGMs is to reduce the computational costs of decoding Forward Error Correction (FEC)-codes, such as the Raptor codes, when using probabilistic methods. In comparison with traditional decoding methods, this approach gains us the power of probabilistic inference while maintaining relatively low complexity.

The first PGM we shall consider is the Bayes Network (BN) in Section 3.3, which represents a joint PD as a product of priors by means of the chain rule. As covered in Section 3.4, the Markov Random Field (MRF) is similar to a BNs, but instead of expanding the PD using the chain rule, we express it as a product of clusters (a.k.a. cliques). In Section 3.5 we find that, for any BN or MRF, there exists an equivalent Factor Graph (FG) that expresses the same conditional independencies as well as the same factorisation of the joint distribution.

Cluster graphs (CGs), as covered in Section 3.6, group all the variables pertaining to a certain cluster in one node. This allows us to propagate PDs over a set of variables, rather than only one, thereby including information on the variables' relationships. Finally, it is possible to construct a CG with a treelike structure from any given PD by means of the Variable Elimination (VE) algorithm. This is known as a Junction Tree (JT), covered in Section 3.7, which allows us to do *exact* inference.

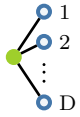
However, before we delve into the application of PGM to FEC-codes, let us first review the basic terminology behind PGMs.

3.2 Basic graph terminology

Here we will only consider the terminology of PGMs that we use in this thesis. Next to each topic in this sub-section we provide a small figure in the right margin as a visual aid to the topic at hand. Although these terminologies are commonly used, we refer to [1, 37] as our sources.

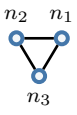
Nodes and edges A node (a.k.a. vertex) is represented either by a circle or a square and contains a PD function. The nodes are connected via lines known as *edges*, which indicate that there exists a dependency between the nodes' PDs. There are two types of edges in graph theory: directed and undirected. A directed edge is depicted with an arrowhead and indicates our understanding of the causal relationship between the PDs of the respective nodes. That is, the node at which the edge is directed is conditionally dependent on the node at its origin. By contrast, an undirected edge indicates joint dependencies between the nodes.

Degree of node ($\mathcal{D}$) The degree of a node is defined by the number of edges connected to it. In Section 2.6 we define the degree of an output node and input in terms of the degree in an MRF rather than its degree in the FG. That is, its degree is equal to the number of other variables connected to it.



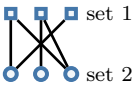
Directed and undirected graphs As the name suggests, directed graphs contain only directed edges and undirected graphs contain only undirected edges. A directed edge indicates a dependency in the direction of the edge, whereas an undirected edge indicates dependencies in both directions. The only type of directed graphs we will consider are BNs.

Cyclic graphs If $\mathcal{N} \geq 3$ represents the number of nodes in a graph, then the graph is a cycle graph if its nodes are connected in a single loop. Moreover, if the graph consists of nodes $\{n_0, n_2, \ldots, n_{\mathcal{N}-1}\}$, then there exists an edge between node n_i and n_{i+1} for $i = 0, 2, \ldots, \mathcal{N}-2$. This chain of nodes becomes a loop with an additional edge between n_0 and $n_{\mathcal{N}-1}$.



Graphs with cycles A graph that contains one or more cycle graph(s) as a sub-graph is said to contain cycles.

Bipartite graphs A PGM that we frequently use is the factor graph. These are bipartite graphs whose nodes can be divided into two disjoint sets, such that every edge connects a node of one set to a node in the other set.



Tree graphs A graph is a tree if, and only if, there exists a unique path between any pair of distinct nodes in the graph, i.e., a tree graph has no cycles. These graphs are the only type over which we are able to do *exact* inference; once a graph contains a cycle we can only approximate our inference.

Leaf nodes A leaf node has a degree $\mathcal{D} = 1$. This terminology pertains to the nature of tree graphs, which are guaranteed to have leaf nodes at the ends of a path along the graph. These nodes play a crucial role in inference, as we will discuss in Chapter 4.

Scope The scope of a function g is represented as $\text{Scp}(g)$ and refers to the set of variables it contains. Thus function $g(x_1, x_3, x_4)$ has a scope $\text{Scp}(g) = \{x_1, x_3, x_4\}$.

Fully connected graphs As the name suggests, a fully connected graph is one where each node in the graph is connected to every other node in the graph. When a fully connected graph appears as a sub-graph, it is often referred to as a cluster.

Neighbours A neighbour of a given node is a node with which it shares an edge.

3.3 Bayes Network

The first PGM we shall consider is the BN. Although we will make little use of this topography, the BN serves as a good introduction to PGMs. For our purpose the general BN represents a joint PD as a product of priors by means of the chain rule,

$$P(\mathbf{x}) = P(x_0)P(x_1|x_0)\dots P(x_{\mathcal{W}}|x_0\dots x_{\mathcal{W}-1}) = \prod_{w=0}^{\mathcal{W}-1} P\Big(x_w|\upsilon(x_w)\Big) \qquad (3.3.1)$$

where the variable set $\mathbf{x}$ has $\mathcal{W}$ variables such that

$$\mathbf{x} = \begin{bmatrix} x_0 & x_1 & \dots & x_w & \dots & x_{\mathcal{W}-1} \end{bmatrix}$$

and $\upsilon(x_i)$ are the dependencies of x_i [38]. However, the chain rule by itself does not reduce the complexity, since the final prior $P(x_{\mathcal{W}}|x_0 \dots x_{\mathcal{W}-1})$ has the same complexity as the original joint PD. To reduce the overall complexity, the independencies need to be taken into account.

As an example, let us use the chain rule to establish a probabilistic equation of the Hamming code with codeword $\mathbf{t} = \mathbf{z}\|\mathbf{p} = z_0z_1z_2z_3p_0p_1p_2$ in order to construct the graph. Our 4 source symbols are independent of one another, and therefore the joint probability of the source symbols is equal to the product of the marginals:

$$P(z_0, z_1, z_2, z_3) = P(z_0)P(z_1)P(z_2)P(z_3).$$

The parity symbols, on the other hand, are all dependent on the source symbols, thus the joint probability of any codeword is defined as:

$$\begin{aligned} P(\mathbf{t}) =& P(z_0)P(z_1)P(z_2)P(z_3)\times \\ & P(p_0|z_0, z_1, z_2)P(p_1|z_1, z_2, z_3)P(p_2|z_0, z_2, z_3). \end{aligned} \tag{3.3.2}$$

Clearly these PDs' complexities are small relative to the joint PD. To elaborate, the complexity of the joint PD is of the order $O(2^{\mathcal{W}})$. Therefore,

$$O_{left}(2^7) \gg O_{right}(2^4).$$

Thus even in such a simple example, factorisation reduces the computational complexity significantly.

The task of depicting (3.3.2) as a BN is simple. For each variable we create a *node*, which represents its respective PD, i.e., the node of p_0 will represent the PD $P(p_0|z_0, z_1, z_2)$. The edges are then added to indicate the dependencies of the variables.

This implies that the source variables, which are independent, will have no edges between each other, whereas the parity symbols will each be connected to 3 of the source symbols. Following this we have the graph in Fig. 3.1. Note that the BN has directed edges, relating to the fact that the parity symbols are causally dependent on the source symbols. Moreover, BNs are directed and acyclic graphs, i.e., there are no cycles in the graph when the edges are followed in their direction [1].

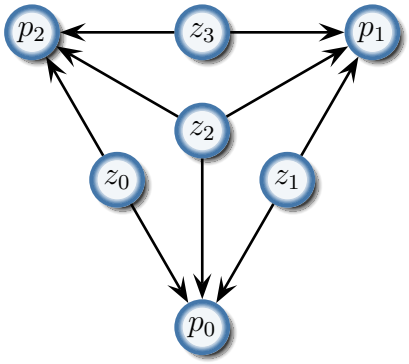


Figure 3.1: The BN of the Hamming code [38], based on (3.3.2). Each node depicts a symbol, each with an associated PD. The BN has directed edges, since each parity symbol is dependent on source symbols, whereas the source symbols are independent.

3.4 Markov Random Field

A MRF is similar to a BNs in its representation of dependencies. However, it takes a different approach in the factorisation of a joint PD. Instead of expanding the PD $P(\mathbf{x})$ using the chain rule, we express it as a product of $\mathcal{W}$ clusters as follows:

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{w=0}^{\mathcal{W}-1} \phi_w(\mathbf{x}_w)$$

with

$$Z = \sum_{\mathbf{x}} \prod_{w=0}^{\mathcal{W}-1} \phi_w(\mathbf{x}_w).$$

Similar to the factors, each cluster $\phi_w(\mathbf{x}_w)$ is a function of a subset of $\mathbf{x}$ such that $\mathbf{x}_w \subset \mathbf{x}$ for $w = 0, 1, \ldots, \mathcal{W}-1$ [39, p. 61]. It is also not required that the clusters be normalised, thus the normalising constant Z is required here as well.

The clusters of an MRF differ from the factors of a BN in that they describe the *joint* PD of the subsets within the distribution instead of the *conditional* PDs. For the graphical representation, this implies two things. Firstly, MRFs are undirected and secondly, all the nodes that form part of the same cluster are fully connected.

Undirected edges increase the possibility that the graph may be cyclic. Despite this, the use of clusters allow for more versatile graph structures since they may be defined arbitrarily. Initially, the clusters do not contain any information, i.e., their respective PDs are uniform. The information of actual

priors needs to be incorporated into these clusters using a rule known as the *family preservation property*.

3.4.1 The family preservation property

For any inference done over the MRF to be correct, it must contain all the available information without redundancy. To achieve this, it is necessary that all the factors of the function be included in the graph by inducing each into one, and only one, cluster. This is known as the family preservation constraint and is formally defined as follows:

Definition 3.1: Family Preservation [1]
For each factor $f_i(\mathbf{x}_i)$, where $i = 0, 1, \ldots, \mathcal{W} - 1$, there exists a cluster $\phi_j(\mathbf{x}_j)$ such that $\mathbf{x}_i \subseteq \mathbf{x}_j$. Furthermore, each factor $f_i(\mathbf{x}_i)$ is assigned to one, and only one, cluster whose scope satisfies $\mathbf{x}_i \subseteq \mathbf{x}_j$.

3.4.2 Hamming code example

To illustrate an MRF we turn to our example of the Hamming code, which may be expressed as follows:

$$P(\mathbf{t}) = \frac{1}{Z}\phi_0(p_0, z_0, z_1, z_2)\phi_1(p_1, z_1, z_2, z_3)\phi_2(p_2, z_0, z_2, z_3) \tag{3.4.1}$$

with Z being the normalising constant in this instance. The resulting MRF is shown in Fig. 3.2.

Through inspection it may be noted that another fully connected cluster $\phi(z_0, z_1, z_2, z_3)$ exists. It is also possible to include smaller clusters, such as $\phi(p_0, z_0, z_2)$ or $\phi(z_1, z_2)$, as long as all the nodes in the cluster are fully connected. This implies that there exist other valid expressions of $P(\mathbf{t})$, such as

$$P(\mathbf{t}) = \frac{1}{Z}\phi_0(p_0, z_0, z_1, z_2)\phi_1(p_1, z_1, z_2, z_3)\phi_2(p_2, z_0, z_2, z_3)\phi_3(z_0, z_1, z_2, z_3).$$

However, this expression is redundant since (3.4.1) satisfies the family preservation property with fewer clusters, leaving the extra clusters empty.

An empty cluster is one that contains a uniform PD, providing no information on how the variables relate to each other, but adding to the complexity of

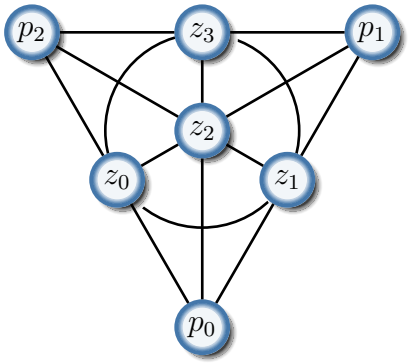


Figure 3.2: The MRF of the Hamming code as based on (3.4.1). The MRF is an undirected graph and consists of multiple clusters, i.e., fully connected sub-graphs of the MRF. Each cluster is defined by a PD stating the pertaining variables' relationship to each other, e.g., $\phi_0(p_0, z_0, z_1, z_2)$ is defined by $P(p_0, z_0, z_1, z_2)$.

the problem. We may therefore, without loss of generality, omit these clusters in order to avoid unnecessary computational costs. For example, consider the cluster $\phi_3(z_0, z_1, z_2, z_3)$. The variables z_0, z_1, z_2, and z_3 are independent and therefore the cluster provides no new information.

3.4.3 The conversion between BNs and MRFs

(a) BN of potential $P(p_0|z_0, z_1, z_2)$

(b) MRF of cluster $\phi_0(p_0, z_0, z_1, z_2)$

Figure 3.3: The conversion of a sub-BN graph to a cluster in an MRF. The act of conversion is called *moralisation* and is done by connecting all the parents (z_0, z_1, and z_2) of the child (p_0) to each other, resulting in a fully connected cluster. This represents the conversion of the conditional PD $P(p_0|z_0, z_1, z_2)$ to the joint PD $\phi_0(p_0, z_0, z_1, z_2)$.

It is possible to convert a BN to an MRF using *moralisation* [39, p. 63]. In doing so, one converts the conditional PDs of the BN to the joint cluster PDs of the MRF by adding edges between all nodes' dependents. This process

may also be done in reverse to convert an MRF to a BN, and is known as *demoralisation*.

For example, the conditional PD $P(p_0|z_0, z_1, z_2)$ in (3.3.2) is depicted in Fig. 3.3(a). Using the chain rule we have

$$P(p_0, z_0, z_1, z_2) = P(z_0)P(z_1)P(z_2)P(p_0|z_0, z_1, z_2).$$

This joint PD may now be allocated to the cluster $\phi_0(p_0, z_0, z_1, z_2)$ as shown in Fig. 3.3(b). If the same procedure is followed for the conditional PDs $P(p_1|z_1, z_2, z_3)$ and $P(p_2|z_0, z_2, z_3)$, we will have successfully converted the BN in Fig. 3.1 to the MRF in Fig. 3.2.

3.5 Factor graphs

BNs and MRFs are both popular graphical models; however, neither model can express all arbitrary factorisations of a joint PD [40]. Fortunately, for any BN or MRF, there exists an equivalent FG that expresses the same conditional independencies and factorisation of the joint distribution.

The FG starts with the same principle as the MRFs by expressing the joint PD as a product. However, instead of clusters, we express the joint as a product of $\mathcal{W}$ *factors*,

$$P(\mathbf{x}) = \frac{1}{Z}\prod_{w=0}^{\mathcal{W}-1} f_w(\mathbf{x}_w). \tag{3.5.1}$$

The initial information contents are the same as before; however, their graphical representation and application differ.

Where a cluster is depicted by the MRF as a mesh of fully connected variable nodes, the FG contains the same information within a single *factor node*. We therefore only need to connect the variable node to their pertaining factors in order to wholly describe the structure of the encoding. We illustrate this in Fig. 3.4, where the factors are represented by the square nodes. The FG of the Hamming code shown here is the graphical representation of the equation

$$P(\mathbf{t}) = f_a(p_0, z_0, z_1, z_2)f_b(p_1, z_1, z_2, z_3)f_c(p_2, z_0, z_2, z_3).$$

These factors only describe the relation of a set of variables and have no direct relationship to each other. This implies that edges only exist between

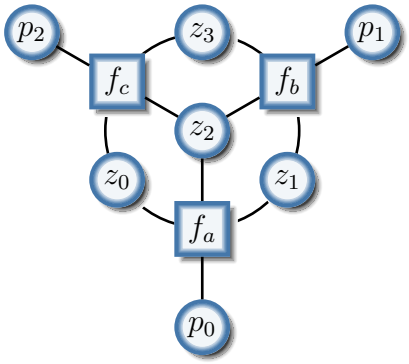


Figure 3.4: The FG of the Hamming code as based on (3.5.1). It is a bipartite graph, having two separate sets consisting of variables and factors respectively. The edges only exist between two nodes of separate sets. The FG is also an undirected graph, allowing inference to flow both ways. The factors describe the dependencies of their pertaining variables, and the variables themselves contain information.

factors and variables. As a result, FGs fall in the group of graphs called *bipartite* graphs, which implies that the graph's nodes can be divided into two disjointed sets such that every edge connects a node of one set to a node in the other set. For FGs, these two sets are the variable nodes and factor nodes. More formally stated, a FG contains an edge between variable x and factor f_w if, and only if, x is an argument of the local function $f_w(\mathbf{x}_w)$, ergo $x \in \mathbf{x}_w$ [37]. Finally, the edges in a FG are always undirected allowing inference to flow both ways.

Of the 3 PGMs we have investigated so far, we will primarily make use of the FG. Our preference towards FGs is due to the fact that factor graphs subsume many other probabilistic graphic models, including MRFs and BNs [41].

3.6 Cluster graphs

Up to this point we have limited our consideration of PGMs to those with single variable nodes. The performance of these types of graphs is limited by the fact that the information passed between the nodes of these structures is only in terms of the particular *single* variables concerned, thereby not explicitly propagating the interaction of the variables.

The topography covered in this section, known as CGs, allows us to include some information of the variable relationships as we propagate the information

across the graph. CGs do this by grouping all the variables pertaining to a given cluster into one node. This often results in many variables repeatedly featuring in multiple nodes as many clusters share sets of variables. To propagate information across the graph, the clusters pass the information they have to the neighbouring clusters with which they share variables. The variables common between any neighbouring clusters are known as the *sepset* of those clusters, and the information passed between the clusters is in the form of a PD with a scope equal to that of the sepset.

We can verify that this approach correlates with our previous PGMs mathematically, following the work done in [39, p. 98]. For simplicity, let us assume that the Hamming code we previously encountered only encodes the first 2 parity bits p_0 and p_1. The joint PD will then be defined as,

$$\begin{aligned} P(z_0, z_1, z_2, z_3, p_0, p_1) &= P(z_0)P(z_1)P(z_2)P(z_3)P(p_0|z_0, z_1, z_2)P(p_1|z_1, z_2, z_3) \\ &= \frac{1}{Z}\phi_0(p_0, z_0, z_1, z_2)\phi_1(p_1, z_1, z_2, z_3). \end{aligned}$$

If we now rearrange the chain rule in (3.3.1) as an expression of the conditional PD such that

$$P(x_{\mathcal{W}}|\upsilon(x_{\mathcal{W}})) = \frac{P(\mathbf{x})}{\prod_{w=0}^{\mathcal{W}-1} P\Big(x_w|\upsilon(x_w)\Big)},$$

we may re-express the 2 clusters that make the joint PD as

$$\begin{aligned} &\frac{1}{Z}\phi_0(p_0, z_0, z_1, z_2)\phi_1(p_1, z_1, z_2, z_3) \\ &= P(z_0)P(z_1)P(z_2)P(z_3)\frac{P(p_0, z_0, z_1, z_2)}{P(z_0)P(z_1)P(z_2)}\frac{P(p_1, z_1, z_2, z_3)}{P(z_1)P(z_2)P(z_3)} \\ &= \frac{P(p_0, z_0, z_1, z_2)P(p_1, z_1, z_2, z_3)}{P(z_1)P(z_2)}. \end{aligned} \tag{3.6.1}$$

From this we assign the clusters to be equivalent to the marginal PDs in the numerator. Accordingly, we may segment the equation to give

$$\begin{aligned} \phi_0(p_0, z_0, z_1, z_2) &= P(p_0, z_0, z_1, z_2), \\ \phi_1(p_1, z_1, z_2, z_3) &= P(p_1, z_1, z_2, z_3), \\ \psi(z_1, z_2) &= Z = P(z_1)P(z_2), \end{aligned}$$

where $\psi(z_1, z_2)$ is referred to as the *separator*. This is because its scope $\{z_1, z_2\} = \{p_0, z_0, z_1, z_2\} \cap \{p_1, z_1, z_2, z_3\}$ is the sepset for those clusters and

therefore $\psi(z_1, z_2)$ describes the mutual dependencies of the clusters. In the light of this we establish a format definition of a cluster graph as provided by David Barber in [39, p. 98]:

Definition 3.2: The Cluster/Clique Graph

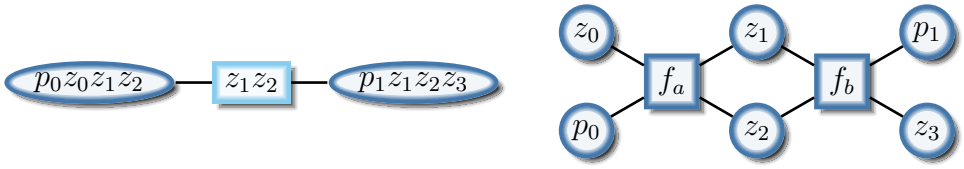
A cluster graph consists of a set of potentials, $\phi_0(\mathbf{x}_0), \ldots, \phi_w(\mathbf{x}_w)$ each defined on a set of variables $\mathbf{x}_w$. For neighbouring clusters on the graph, defined on sets of variables $\mathbf{x}_i$ and $\mathbf{x}_j$, the intersection $\mathbf{x}_s = \mathbf{x}_i \cap \mathbf{x}_j$ is called the *separator* and has a corresponding potential $\psi_s(\mathbf{x}_s)$. A cluster graph represents the function

$$P(\mathbf{x}) = \frac{\prod_w \phi_w(\mathbf{x}_w)}{\prod_s \psi_s(\mathbf{x}_s)}, \tag{3.6.2}$$

where $\mathbf{x}_w$ and $\mathbf{x}_s$ are sepsets of $\mathbf{x}$.

Figure 3.5 shows the equivalent CG in Fig. 3.5(a) and FG in Fig. 3.5(b) for the equation

$$P(z_0, z_1, z_2, z_3, p_0, p_1) = \frac{\phi_0(p_0, z_0, z_1, z_2)\phi_1(p_1, z_1, z_2, z_3)}{\psi(z_1, z_2)}.$$



(a) Cluster graph (b) Factor Graph

Figure 3.5: A comparison of the representation of 2 clusters as a CG and its equivalent FG. From this it is clear that the conversion between these 2 graphs, and subsequently BNs and MRFs, can be done by inspection alone.

By using this definition we may easily construct the CG of a joint by simply identifying the clusters, either mathematically or from other graphs, and connecting each cluster node with any other that shares one or more variables with it. Thus the fully constructed CG of the Hamming code can be found in Fig. 3.6.

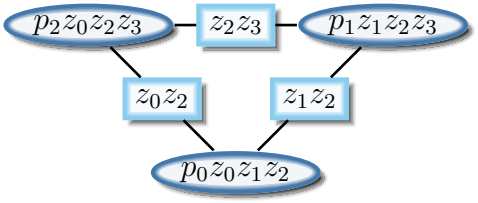


Figure 3.6: The CG of the Hamming code, converted from Fig. 3.4. Note that this graph does not satisfy the running intersection property as explained in Section 3.6.1.

However, when using this method we must ensure that our newly constructed CG adheres to two constraints in order to be a valid representation of the joint PD. The first is the family preservation property as explained in Section 3.4.1 and the second is the Running Intersection Property (RIP).

3.6.1 The running intersection property

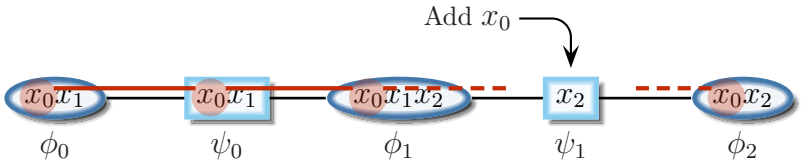
The RIP ensures that the flow of inference over the CG is properly functional for each variable contained in the graph. Its formal definition is as follows:

Definition 3.3: Running Intersection Property [1]
For each cluster pair $\phi_i(\mathbf{x}_i)$, $\phi_j(\mathbf{x}_j)$ and variable $x \in \mathbf{x}_i \cap \mathbf{x}_j$ there exists a *unique* path between $\phi_i(\mathbf{x}_i)$ and $\phi_j(\mathbf{x}_j)$ for which all clusters and sepsets contain x.

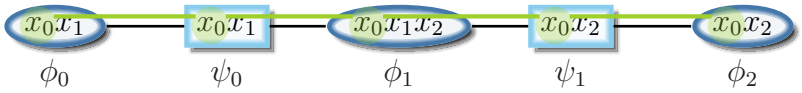
There are two possible ways in which a CG may fail to adhere to this property, either by containing isolated clusters in regard to a particular variable or containing paths with loops.

A simplistic depiction of the first is given in Fig. 3.7, where the RIP of x_0 is disjointed at sepset $\psi_1(x_2)$, separating $\phi_2(x_0, x_2)$ from the rest of the graph information on x_0. This case may result in the development of two different beliefs on x_0 after the message-passing algorithm terminates. Not only may these beliefs be conflicting, but they will be suboptimal since they do not take all the available information into account. We therefore need to add x_0 to the scope of $\psi_1(x_2)$ in order to satisfy the RIP.

For the approach to constructing a CG that we consider in this work, the disjointing of variables is not a concern. However, the chance to violate the RIP by paths containing loops is far more likely. A loop in the path of a



(a) Path of inference for x_0

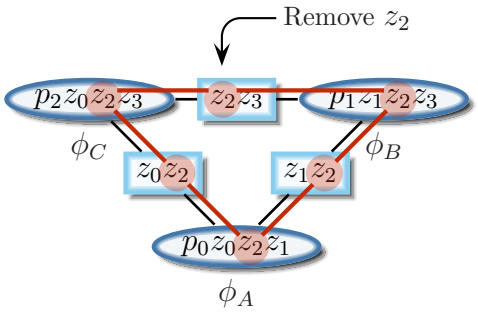


(b) CG satisfies RIP

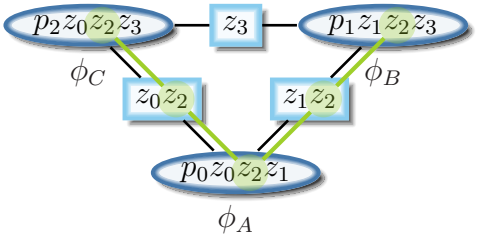
Figure 3.7: An arbitrary chain CG depicting a scenario where the RIP is violated. The violation is due to the 2 disjointed paths of inference for x_1. This may be rectified by adding x_0 to separator $\psi_1(x_2)$.

variable will cause the initial information of a cluster to feed back on itself as it is propagated through the graph, resulting in a "biased opinion" on the state of the variable.

Such an example is found in the CG of the Hamming code that we constructed in Fig. 3.6. In Fig. 3.8(a) we show the path of inference for z_2, which is a loop. To correct this we may simply remove z_2 from any one of the sepsets to satisfy the RIP, as can be seen in Fig. 3.8(b).



(a) Path's of inference for z_0 and z_2



(b) CG satisfies RIP

Figure 3.8: The preservation of the RIP on the Hamming code CG. In (a) the path of inference for z_2 is highlighted in red since it does not form a *unique* path and therefore violates the RIP. The variable z_2 is thus removed from one of the sepsets in order to comply with the RIP as in (b).

This may be verified mathematically by re-expression of the joint PD of

the Hamming code according to (3.6.2):

$$\begin{aligned} P(\mathbf{t}) &= P(z_0)P(z_1)P(z_2)P(z_3)P(p_0|z_0,z_1,z_2)P(p_1|z_1,z_2,z_3)P(p_2|z_0,z_2,z_3) \\ &= \frac{P(p_0,z_0,z_1,z_2)P(p_1,z_1,z_2,z_3)P(p_2,z_0,z_2,z_3)}{P(z_0)P(z_1)P(z_2)^2P(z_3)} \\ &= \frac{\phi_A(p_0,z_0,z_1,z_2)\phi_B(p_1,z_1,z_2,z_3)\phi_C(p_2,z_0,z_2,z_3)}{\psi_{AC}(z_0,z_2)\psi_{AB}(z_1,z_2)\psi_{BC}(z_3)}. \end{aligned}$$

Note that the denominator in the 2nd step is a product of 5 marginals. We choose the scope of the 3 PDs in the numerator as the scope of the resulting clusters. Furthermore, we group the PDs in the denominator into sepsets of the clusters. The grouping of the sepsets may be done in any order, thus we group $P(z_0)$ and one of $P(z_2)$'s together to form $\psi_0(z_0,z_2)$ as the separator between clusters $\phi_0(p_0,z_0,z_1,z_2)$ and $\phi_1(p_1,z_1,z_2,z_3)$. Likewise we group $P(z_1)$ and the other $P(z_2)$ together as the separator between $\phi_0(p_0,z_0,z_1,z_2)$ and $\phi_2(p_2,z_0,z_2,z_3)$. This leaves only $P(z_3)$ in the denominator for the final separator between $\phi_1(p_1,z_1,z_2,z_3)$ and $\phi_2(p_2,z_0,z_2,z_3)$.

3.7 Junction trees

The previous four PGMs all result in loopy graphs when applied to the FECs. As we mentioned in Section 3.2, cycles are not desirable due to their biased nature, which limits us to *approximate* inference. We will now consider a method that uses Variable Elimination (VE) to convert these graphs to a CG with a treelike structure, called a JT. The treelike structure of the JT will then allow us to do *exact* inference [1].

3.7.1 JT construction using VE

The VE algorithm is one of the simplest and most fundamental algorithms for inference over graphical models. In this section we focus on how VE can be used to construct a JTs and in Chapter 4 the implications of VE toward inference are considered.

To use the VE algorithm to construct a new graph, we repeatedly eliminate one variable from the given function. For every elimination, we add a cluster to the graph, of which the scope is the union of the scopes of the clusters that includes the eliminated variable. For example, if clusters $\phi_0(\mathbf{x}_0)$ and $\phi_1(\mathbf{x}_1)$

are given, and $x_e \in \mathbf{x}_0$ as well as $x_e \in \mathbf{x}_1$, the elimination of x_e will result in a cluster $\phi_G(\mathbf{x}_G)$ added to the graph with $\text{Scp}(\phi_G) = \mathbf{x}_0 \cup \mathbf{x}_1$. This is repeated until all variables have been removed and we are left with a function with an empty scope.

The function itself is altered for each elimination, in that all clusters that contained the eliminated variable are replaced with a sepset. This sepset will be added to the graph along with the newly generated cluster. The scope of the sepset is equal to that of the cluster, excluding the variable that was eliminated. That is,

$$\text{Scp}(\psi_G) = \text{Scp}(\phi_G) \setminus x_e,$$

where the symbol "\" denotes a set exclusion. This sepset is connected to the cluster that resulted from the said elimination. The other end of the sepset will be connected to the cluster that is formed from the elimination of a variable in its scope.

We again use the Hamming code as a practical way to illustrate this process. As we already know, the Hamming code is defined by 3 factors, i.e.,

$$P(\mathbf{t}) = f_0(p_0, z_0, z_1, z_2) f_1(p_1, z_1, z_2, z_3) f_2(p_2, z_0, z_2, z_3).$$

The order in which we eliminate the variables is $\langle p_0, p_1, p_2, z_3, z_2, z_1, z_0 \rangle$. The reason will be explained shortly. Fig. 3.9 shows the resulting JT. The reader is encouraged to refer to this image at each step. However, down-scaled versions are given in the right margin next to each step, as a visual aid. In each of these visual aids the solid (blue) nodes are the ones involved in the current step. The dashed (grey) nodes have not yet been constructed and those with solid lines were previous additions to the JT.

The first 3 eliminations, $\langle p_0, p_1, p_2 \rangle$, are independent since each variable is contained in separate cluster scopes. We therefore do them simultaneously so that

$$\begin{aligned} P(z_3, z_2, z_1, z_0) &\sim f_0(\cancel{p_0}, z_0, z_1, z_2) f_1(\cancel{p_1}, z_1, z_2, z_3) f_2(\cancel{p_2}, z_0, z_2, z_3) \\ &\sim \psi_0(z_0, z_1, z_2) \psi_1(z_1, z_2, z_3) \psi_2(z_0, z_2, z_3). \end{aligned}$$

This forms the 1st stage of the graph, consisting of the following 3 cluster and

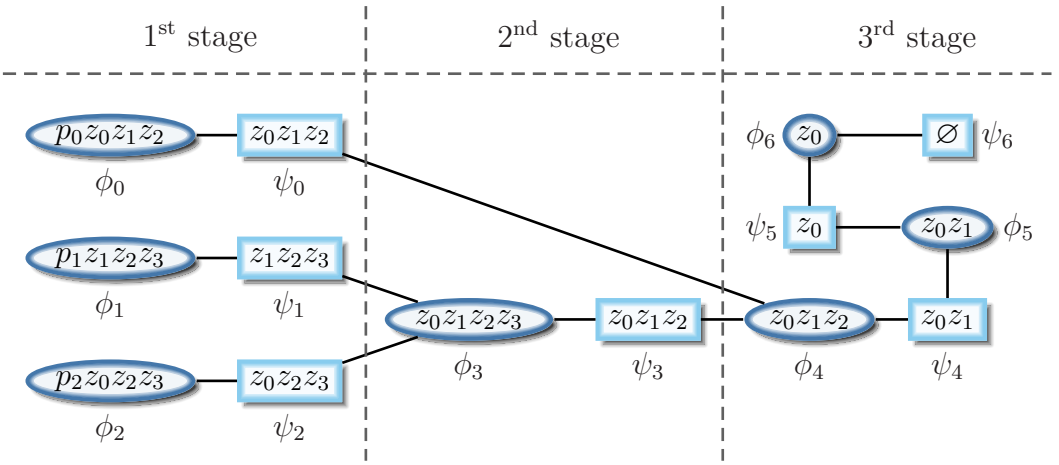


Figure 3.9: The resulting JT of the Hamming code constructed using the VE algorithm with elimination order $\langle p_0, p_1, p_2, z_3, z_2, z_1, z_0\rangle$. It is divided into 3 stages of construction. Clusters $\phi_i(\mathbf{z}_i)$ and $\psi_i(\mathbf{z}_i)$ are redundant for $i = 4, 5, 6$ and may be absorbed to obtain the more effective JT in Fig. 3.10.

sepset pairs:

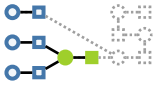
$$\begin{aligned}&\phi_0(p_0, z_0, z_1, z_2) \text{ and } \psi_0(z_0, z_1, z_2),\\ &\phi_1(p_1, z_1, z_2, z_3) \text{ and } \psi_1(z_1, z_2, z_3),\\ &\phi_2(p_2, z_0, z_2, z_3) \text{ and } \psi_2(z_0, z_2, z_3).\end{aligned}$$

The following elimination is z_3:

$$\begin{aligned}P(z_2, z_1, z_0) &\sim \psi_0(z_0, z_1, z_2)\psi_1(z_1, z_2, \cancel{z_3})\psi_2(z_0, z_2, \cancel{z_3})\\ &\sim \psi_0(z_0, z_1, z_2)\psi_3(z_0, z_1, z_2),\end{aligned}$$

which form the cluster and sepset pair

$$\phi_3(z_0, z_1, z_2, z_3) \text{ and } \psi_3(z_0, z_1, z_2).$$



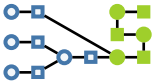
Here we have two clusters in the function that include the eliminated variable in their scopes. As a result, both these sepsets are connected to the newly formed cluster $\phi_3(z_0, z_1, z_2, z_3)$ in the 2nd stage of the graph.

The final 3 eliminations $\langle z_2, z_1, z_0\rangle$ follow the same logic:

$$\begin{aligned}P(z_1, z_0) &\sim \psi_0(z_0, z_1, \cancel{z_2})\psi_3(z_0, z_1, \cancel{z_2})\\ P(z_0) &\sim \psi_4(z_0, \cancel{z_1})\\ P(\varnothing) &\sim \psi_5(\cancel{z_0})\\ &\sim \psi_6(\varnothing).\end{aligned}$$

This results in the 3rd and final stage of the graph, with the clusters and sepsets

$$\phi_4(z_0, z_1, z_2) \text{ and } \psi_4(z_0, z_1),$$
$$\phi_5(z_0, z_1) \text{ and } \psi_5(z_0),$$
$$\phi_6(z_0) \text{ and } \psi_6(\varnothing).$$



3.7.2 Absorption of redundant clusters

As mentioned before, a cluster *fully* describes the relationship between all the variables in its scope. Therefore, any cluster is redundant where its scope is a subset of a neighbouring cluster and results in unnecessary additional computations. This is the case with our construction of the JT for the Hamming code in Fig. 3.9. Note that

$$\text{Scp}(\phi_6) \subset \text{Scp}(\phi_5) \subset \text{Scp}(\phi_4) \subset \text{Scp}(\phi_0).$$

We may remove this redundancy by simply absorbing the clusters $\phi_6(z_0)$, $\phi_5(z_0, z_1)$ and $\phi_4(z_0, z_1, z_2)$ into $\phi_0(p_0, z_0, z_1, z_2)$, resulting in the more effective JT in Fig. 3.10.

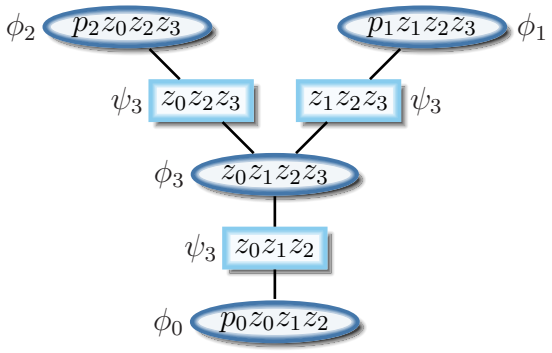


Figure 3.10: The compact version of the JT in Fig. 3.9. This graph has the lowest complexity of $O(2^4)$ possible for a JT of the Hamming code. With as little redundancy as possible, this is the ideal graph for exact inference.

This absorption allows us to end the VE algorithm as soon as all the remaining variables are contained in the scope of the cluster generated by the previous elimination, as all the remaining variables' potential clusters will be absorbed by this cluster.

3.7.3 Properties of a JT constructed from VE [1]

VE will *always* result in a JT. The reason for this is due to the removal of each cluster or sepset from the function once one of the variables in its scope has been eliminated. As a result, no cluster or sepset is used more than once throughout the VE algorithm. This implies that there exists only one sepset for each cluster and it is inherently impossible to construct a non-treelike graph with $\mathcal{W}$ clusters and $\mathcal{W}-1$ sepsets, assuming the graph is not disjointed. Thus the resulting CG must be treelike, i.e., a JT.

The JT is also *family preserving*. This is inherently so, because we start the VE algorithm using the priors of the joint PD. Thus the initial clusters' scopes will be equal to that of the factors and will be able to contain those factors.

Finally, the resulting JT will satisfy the RIP. We can guarantee this, since for each elimination of a variable in the VE algorithm, all instances of that variable are eliminated. In other words, for the elimination of variable x from the function, all clusters and sepsets that contain x in their scope are also removed from the function. This implies that if a variable is contained in a cluster, it must be present in all other clusters along a path, up to the point where it is eliminated. Thereafter, no other cluster can contain the variable.

3.7.4 The importance of the order of elimination

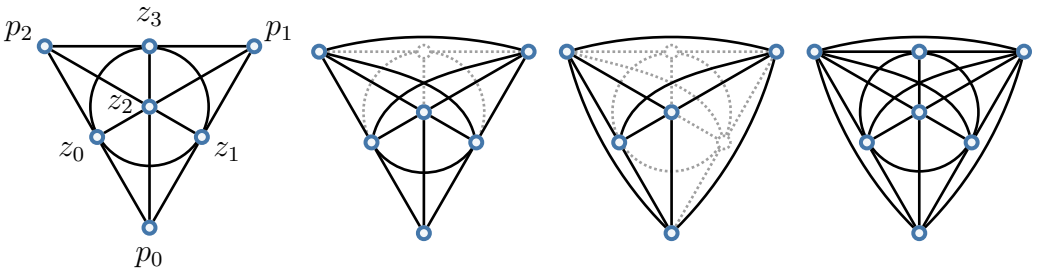
The VE algorithm may have any elimination order, however it is more efficient to select the elimination order carefully. The wrong elimination order will result in a suboptimal JT, with a higher computational complexity than other JTs possible for the same function.

The order of complexity of the graph is measured by the size of its largest cluster. For example, the resulting JT of the Hamming code constructed with the elimination order $\langle z_3, z_1, z_0, z_2, p_0, p_1, p_2 \rangle$ has a complexity of $O(2^6)$. It is depicted in Fig. 3.11 after the redundant cluster has been absorbed. This graph is sub-optimal since it has a greater complexity when compared to the graph in Fig. 3.10, which has a complexity of only $O(2^4)$.

Unfortunately, finding the optimal elimination order remains a Non-deterministic Polynomial-time hard (NP-hard) problem [39, p. 418]. An efficient approximation method often used involves applying the VE algorithm to the given function's MRF.

Figure 3.11: The resulting JT after the VE with order $\{z_3, z_1, z_0, z_2, p_0, p_1, p_2\}$ for the Hamming code. The complexity of this graph is $O(2^6)$, which is poor compared to the $O(2^4)$ complexity of the JT in Fig. 3.10.

For it to be possible to convert an MRF into a JT, it must describe the function at every stage of the VE algorithm. We also know that a cluster is represented by a fully connected sub-graph within an MRF. Thus if any of the clusters formed by the algorithm are not described by the MRF, we must add the necessary edges, called *induced edges*, to the original graph. These edges are added between any pair of nodes that are connected to the eliminated variable, but not to each other.



(a) Original MRF (b) z_3 eliminated (c) Fully connected graph (d) Resulting MRF

Figure 3.12: A non-ideal VE sequence for the Hamming code, with elimination order $\{z_3, z_1, z_0, z_2, p_0, p_1, p_2\}$. The initial MRF of Fig. 3.2 is shown in (a). (b) shows the graph after the elimination of variable z_3. In (c) we find that the graph is fully connected, thus further elimination is not necessary. (d) depicts the final graph after the algorithm was terminated.

Figure 3.12 shows this approach for the elimination order used to obtain the JT in Fig. 3.11. The initial graph is depicted in Fig. 3.12(a) and after the first elimination (z_3), 3 edges are induced between node pairs (z_0, p_1), (z_1, z_2), and (p_1, p_2), as shown in Fig. 3.12(b). Variable z_3 and its respective edges no longer form part of the MRF at this stage. However, we leave it depicted in grey to make the steps of the VE easier to follow. The next elimination, that of variable z_1 in Fig. 3.12(c), produces 2 more induced edges between node pairs (p_0, p_1) and (p_0, p_2).

At this point it may be noticed that the remaining graph is fully connected. This implies that the remainder of the VE algorithm is irrelevant, since no more edges can be induced. Thus the resulting MRF will be as shown in Fig. 3.12(d).

By investigating how VE changes the structure of the MRF for any elimination order, we may estimate the cost of that order by using a heuristic cost function. This is often done using the *greedy search* algorithm, where the decision of which variable to eliminate is done *on-the-fly*. That is, the decision is based solely on the cost of the eliminated variable and not the ramifications it may have at a later stage in the algorithm. Even though this approach does not guarantee the best VE order, it has proved to deliver sufficient results while retaining low complexity [1, p. 314].

The cost functions may be defined as any of the following:

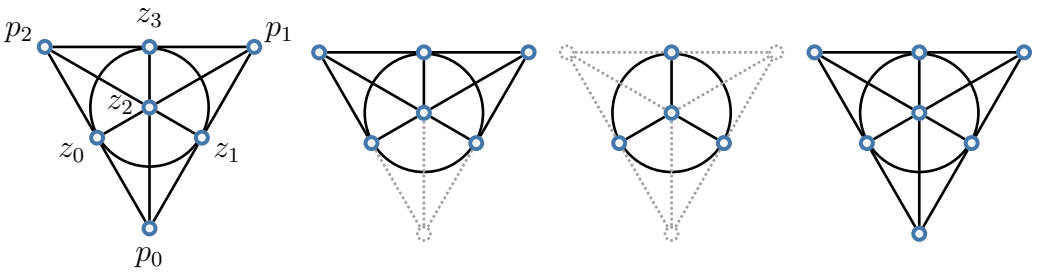
- *min-neighbours*: An elimination cost is based on the number of neighbours the respective node currently has in the graph.
- *min-weight*: The cost is based on the cardinality ($|X|$) of the PD that is associated with the resulting cluster due to the elimination.
- *min-fill*: The number of induced edges of each elimination defines the cost.
- *weighted min-fill*: The cost is the total weight of the induced edges, where an edge weight is the product of cardinality of the 2 nodes they connect.

If we now consider the elimination order $\langle p_0, p_1, p_2, z_3, z_2, z_1, z_0 \rangle$ we used for the optimal JT in Fig. 3.10, we find that it does not induce any edges as shown in Fig. 3.13[1]. Because the minimal edges were induced, we may conclude that it is the JT with the lowest complexity possible for the Hamming code.

3.8 Summary

The application and depiction of PDs by means of PGMs is covered in this chapter. We also see how using PGMs reduces the complexity of a probabilistic inference problem. This chapter does not, however, cover probabilistic inference, which is left for Chapter 4.

[1] An optimal elimination order does not necessarily produce no induced edges



(a) Original MRF (b) p_0 eliminated (c) Fully connected graph (d) Resulting MRF

Figure 3.13: The ideal VE sequence for the Hamming code, with elimination order $\{p_0, p_1, p_2, z_3, z_2, z_1, z_0\}$. Again, the initial MRF of Fig. 3.2 is shown in (a). (b) shows the graph after the elimination of variable p_0. In (c), after the further elimination of p_1 and p_2, we find that the graph is fully connected, thus further elimination is not necessary. No edges were induced using this elimination order and it is thus superior to that of Fig. 3.12.

In Section 3.2 the necessary basic graph terminology is covered, such as defining the degree of a node or a cyclic graph, giving the reader the necessary background terminology for the work ahead.

The first PGM we consider is the BN in Section 3.3. For our purpose the general BN represents a joint PD as a product of priors by means of the chain rule in (3.3.1), while taking all independencies into account. Even with a simplistic example such as the Hamming code, we observe a reduction in complexity.

As covered in Section 3.4, the MRF is similar to a BNs, but instead of expanding the PD using the chain rule, we express it as a product of clusters (aka cliques). The information of actual priors is then incorporated into the clusters, which is known as the *family preservation property* as given in Definition 3.1. Even though this approach opens up the possibility for the presence of cycles within the graph, it allows for more versatile graph structures. It is also possible to convert a BN to its equivalent MRF and *vice versa*.

In Section 3.5 we find that, for any BN or MRF, there exists an equivalent FG that expresses the same conditional independencies as well as the same factorisation of the join distribution. FGs achieve this by containing the information of a cluster within a single *factor node*. As a result they are undirected *bipartite* graphs with edges only between factors and their corresponding variables.

CGs, as covered in Section 3.6, improve on the previous graph types by

allowing us to include some information of the variables' relationships as we propagate the information across the graph. It does this by grouping all the variables pertaining to a certain cluster into one node, allowing us to propagate PDs over a set of variables, rather than only one, thereby including information on the variables' relationships. However, we must ensure that our CGs adhere to both the *family preservation property* and the Running Intersection Property in order to be a valid representation of the joint PD.

Finally, it is possible to construct a CG with a treelike structure from any given PD by means of the VE algorithm. This is known as a JT, covered in Section 3.7, which allows us to do *exact* inference. A JT constructed using the VE algorithm is guaranteed to be *family preserving* as well as to satisfy the RIP. However, the order of elimination while constructing the JT is important, since the wrong elimination order will result in a JT with a higher computational complexity than other possible JTs for the same function. Unfortunately, finding the optimal elimination order remains a NP-hard problem. An efficient approximation method often used involves applying the VE algorithm, along with some heuristic cost function, to the given function's MRF.

Chapter 4

Belief propagation

4.1 Introduction

Probabilistic Graphical Models (PGMs) serve as a useful tool to depict the relationship and inter-dependencies of a set of variables. However, their true potential comes to fruition once inference is applied to them. In Section 4.2 we consider the message-passing principles that are applied to most of these probabilistic models.

In Section 4.3 we cover the use of the Variable Elimination (VE) algorithm to reduce the complexity of inference by dividing the process into multiple steps. The update of the remaining potentials can be considered as the passing of a message to the neighbouring node(s).

Section 4.4 gives a qualitative description of how Belief Propagation (BP) extends the concept of message-passing in order to apply inference over PGMs. Also known as the sum-product algorithm, it involves only the two operations from which the name is derived, i.e., summations and products.

In Section 4.5 we consider only three of the more efficient and widely-used schedules, namely:

- Standard forward-backward execution
- Message flooding (synchronous BP)
- Sequential message-passing (asynchronous BP).

For treelike graphs, the algorithms are guaranteed to converge to an exact solution. This is not the case with graphs containing cycles and yet it remains

of practical importance as it often does converge to an approximate solution.

The *Tanh-rule* in Section 4.6 reduces the complexity of the BP from $O(2^N)$ to $O(N)$, where N is the number of variables in the graph. However, this approach requires that the messages be confined to only a single Binary Random Variable (BRV), which severely limits its usefulness.

BP over Cluster graphs (CGs) may be considered to be a more generalised form of BP, as explained in Section 4.7. The advantage of this more generalised form of BP lies in the fact that these messages' scopes are not restricted to a single variable. This allows for more information to be propagated for every message passed. Thus, on average, a CG will converge in fewer iterations than its equivalent Factor Graph (FG).

4.2 Fundamentals of message-passing

The concept of message-passing is derived from the idea that the collective solutions to the local parts form the solution of the global problem. To illustrate this principle, we will use a similar example to that presented by David J.C. MacKay in [2, p. 241]. In our example there exists a network consisting of one server and multiple nodes. The task of the server is to establish how many nodes there are in the network at any point in time, given the constraint that only nodes that are directly linked can communicate with one another.

As an introductory example, let us assume that the nodes are connected in a single line as shown in Fig. 4.1. The server is represented by the node denoted with the letter 's' and all other nodes with the letter 'n'.

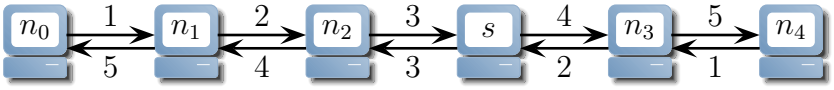


Figure 4.1: The graph depicts a network coupled together to form a line and maintained by a single server (marked as s). A count (message) is passed from left to right (n_0 to n_4) and *vica versa*. At each pass the counts are incremented by 1. To obtain the total number of nodes in the network, the server adds the two messages it receives together.

Since the server can only communicate with nodes n_2 and n_3, both of them must be able to share information about the number of nodes that are further down the line to the server. Similarly, these nodes can only communicate with

the server and one more node down the line. This pattern continues to both ends of the line. Therefore, in order for the server to be able to determine the number of nodes, it is necessary to start the algorithm at both ends of the line, i.e., n_0 and n_4. These nodes play a pivotal role in the message-passing process, and are accordingly denoted as *leaf nodes*.

Leaf nodes count themselves as 1 and respectively pass their count on to their neighbouring node. Thereafter, whenever a node receives a count (message), it adds 1 to the count and passes it on to the next node. If this process continues until both counts reach the server, it can add the results together to calculate the total nodes in the network. In the example given in Fig. 4.1, this will be $2 + 3 = 5$. Not only does the server now have the total number of nodes, but also their respective *positions* within the network.

Realising that this network example may be closely associated with a PGM, we may deduce a few properties of the message-passing algorithm. Firstly, it has an order of complexity proportional to the number of nodes and, secondly, each message contains *all* the information of the nodes preceding it.[1] This implies that, after termination of the algorithm, *all* the nodes in the network have access to the total information contained in the system. Finally, the algorithm must start at the leaf nodes in order to be able to propagate all available information.

This message-passing scheme works for any network that has a treelike graph structure such as the example given in Fig. 4.2. However, since each message must contain *all* the information of the nodes preceding it, each node must wait until it has received a message from all of its neighbouring nodes *except* the node that the message will be passed on to. Only once all the counts have been received is the total calculated and the result passed to the recipient node.

For example, if we denote a message passed from node n_i to node n_j as $\mu_{n_i \to n_j}$, then the message passed from node n_6 to the server is calculated as

$$\mu_{n_6 \to s} = \mu_{n_1 \to n_6} + \mu_{n_5 \to n_6} + 1 = 2 + 1 + 1 = 4$$

For the server to calculate the total nodes in the network, it simply needs to add up all the message values it receives and to include itself:

$$\mu_{n_6 \to s} + \mu_{n_7 \to s} + 1 = 4 + 5 + 1 = 10$$

[1] In the network examples, the information contained by each node is '1'

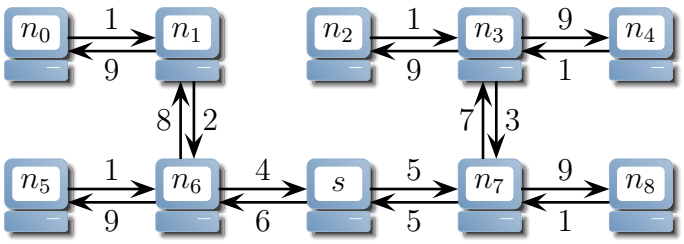


Figure 4.2: The graph depicts an extended network connected in a treelike manner and maintained by a single server (marked as s). A count (message) is initiated at each leaf node and propagated across the network. At each pass the counts are incremented by 1. By adding all of the incoming messages at any node of the network, the total number of nodes in the network will be obtained.

As the examples above illustrate, in order to obtain an exact solution, it is important that each node must receive the information residing within all other nodes. Also, each node must only receive the information exactly once, otherwise the information will be duplicated and the final result skewed. Unfortunately, in the case of graphs with loops, this is not possible to guarantee and thus it is impossible to guarantee an exact solution as an end result.

4.3 The Variable Elimination algorithm

In Section 3.7.1 we covered the use of the VE algorithm to construct a Junction Tree (JT). By following the work done in [39, p. 75], we now focus on how this algorithm reduces the computational complexity of inference by dividing the process into multiple steps.

Consider the joint Probability Distribution (PD) that has independencies such that $P(x_0, x_1, x_2, x_3) = P(x_0, x_1)P(x_1, x_2)P(x_2, x_3)$ and say we wish to calculate the marginal PD of variable x_0. To do so, we must sum over all the other variables:

$$\begin{aligned} P(x_0) &= \sum_{x_1,x_2,x_3} P(x_0, x_1, x_2, x_3) \\ &= \sum_{x_1,x_2,x_3} P(x_0, x_1)P(x_1, x_2)P(x_2, x_3). \end{aligned} \tag{4.3.1}$$

However, the scopes of potential $P(x_0, x_1)$ and $P(x_1, x_2)$ do not contain variable x_3, nor does the scope of $P(x_0, x_1)$ contain x_2. Consequently, these potentials do not change when these variables are summed out. This allows us to move them to the outside of the respective summations by means of the

distributive law, such that

$$P(x_0) = \sum_{x_1} P(x_0, x_1) \sum_{x_2} P(x_1, x_2) \sum_{x_3} P(x_2, x_3). \tag{4.3.2}$$

This separation of the summation is known as the VE algorithm, since each summation eliminates one variable from the function. If we assume that all the variables are BRVs, a total of $3 \times 2^1 = 6$ addition operations are required to calculate the marginal. By contrast, the naive marginalisation approach in (4.3.1) needs $2^4 - 2 = 14$ addition operations.[2]

Even though this difference in terms of the reduction in computational complexity is small, the VE algorithm's computational complexity scale is proportional to $O(2^{N-1})$, where N is the number of variables in the largest cluster. This is in contrast to the computational complexity scaling of the naive approach that is proportional to $O(2^M)$, where M is the number of variables in the joint PD. Since N is guaranteed to be smaller than M, VE is guaranteed to reduce the computational complexity.

For each elimination, the VE algorithm changes the structure as well as the potentials of the Markov Random Field (MRF) of the above PD, as shown in Fig. 4.3. Here the elimination is illustrated by the removal of its respective node and the possible induction of new connections, as covered in Section 3.7.4. The update of the remaining potentials can be considered as the passing of a message to the neighbouring node(s).

(a) Initial MRF of $P(x_0, x_1, x_2, x_3)$ (b) Marginal potential as message

Figure 4.3: The elimination of variable x_3 in the MRF of the PD in (4.3.2). (a) shows the original graph and (b) shows the elimination of the variable by the removal of its node and the updating of its neighbouring node(s) by passing of a message.

For example, the message sent from node x_3 to node x_2 is

$$\mu_{x_3 \to x_2}(x_2) = \sum_{x_3} P(x_2, x_3).$$

[2]This does not include the multiplication operations, which also see a reduction in computational complexity in the case of VE

Accordingly, the potential over nodes x_1 and x_2 becomes $P(x_1, x_2)\mu_{x_3 \to x_2}(x_2)$ in order to satisfy the updated version of (4.3.2):

$$P(x_0) = \sum_{x_1} P(x_0, x_1) \sum_{x_2} P(x_1, x_2)\mu_{x_3 \to x_2}(x_2).$$

4.4 The sum-product algorithm

BP extends the concept of message-passing and the principles of VE in order to apply inference for all variables in a given PGM. It is also known as the sum-product algorithm, involving only the two operations from which the name is derived, i.e., summations and products.

This section gives a qualitative description of BP, where a more formal definition may be found in [39, p. 578]. Since both Bayes Networks (BNs) and MRFs can be represented as an FG, it is convenient to focus on application of the algorithm towards FGs as it subsumes both the others.

Since there are two types of nodes in an FG, the sum-product involves two types of messages for these type of graphs: those from factor node to variable nodes, and vice versa, as shown in Fig. 4.4.

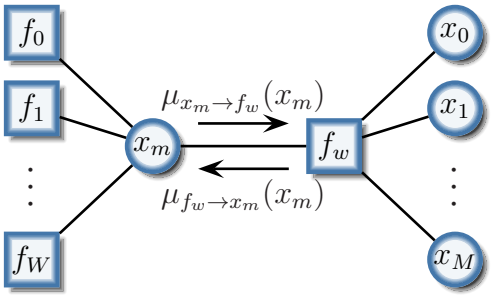


Figure 4.4: The sub-graph of an arbitrary FG showing the two types of messages involved in the sum-product algorithm. Note that the scope of both messages consists solely of the variable at hand (x_m).

As explained in Section 3.5, the factor nodes of the graph are representations of the factors in the posterior PD. Thus, each factor contains partial information on the likelihoods of the variables in its scope. The message sent from a factor may accordingly be viewed as its *conjecture* on the PD of the receiving variable. Thus, the message from factor node f_w to variable node x_m is $\mu_{f_w \to x_m}(x_m)$ is dependant on $f_w(\mathbf{x}_w)$, where $\mathbf{x}_w \subseteq \{x_0, x_1, \ldots, x_m, \ldots, x_\mathcal{M}\}$.

However, as in the case of the network in Section 4.2, it is necessary to include the information residing within all the nodes proceeding f_w from the point of view of x_m. Thus the factor must collect all the messages from its neighbouring nodes and include these in its conjecture. This excludes the message of the receiving variable itself, since it already possesses this information and adding it to the message will duplicate said information, creating a bias towards it. Subsequently, the message $\mu_{f_w \to x_m}(x_m)$ is a function of $q(\mathbf{x}_w)$, where

$$q(\mathbf{x}_w) = f_w(\mathbf{x}_w) \prod_{\substack{x_i = \\ \mathbf{x}_w \setminus x_m}} \mu_{x_i \to f_w}(x_i).$$

The subscript of the product $x_i = \mathbf{x}_w \setminus x_m$ implies that x_i is equal to each element in $\mathbf{x}_w$ for the sequence of operations, except for x_m.

Since the message $\mu_{f_w \to x_m}(x_m)$ is only in terms of x_m, the function $q(\mathbf{x}_w)$ is accordingly summed over all other variables, except x_m, to obtain a marginal function of the variable such that

$$\mu_{f_m \to x_m}(x_m) = \sum_{\substack{x_i = \\ \mathbf{x}_m \setminus x_m}} \Big(q(\mathbf{x}_w) \Big).$$

By expanding this equation the final message is obtained as given in Definition 4.1 by equation (4.4.1).

Unlike factor f_w, variable x_i contains no initial information and thus we have no knowledge of it beyond what can be derived from its relationships with the other variables and any information pertaining to them. Since this information is contained by its neighbouring factors, the message from the variable x_i to factor f_j is simply the combination of their conjectures of its PD. This is done as a product of the conjectures, as given in equation (4.4.2) in Definition 4.1. For the same reason as with $\mu_{f_w \to x_m}(x_m)$, this message

excludes the conjecture of factor f_j itself.

Definition 4.1: The FG messages of BP [2, p. 336] [41, 42]
Updating rule for factor to variable:

$$\mu_{f_m \to x_n}(x_n) = \sum_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \left(f_m(\mathbf{x}_m) \prod_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \mu_{x_i \to f_m}(x_i) \right). \tag{4.4.1}$$

Updating rule for variable to factor:

$$\mu_{x_n \to f_m}(x_n) = \prod_{\substack{f_i = \\ \mathbf{f}_n \setminus f_m}} \mu_{f_i \to x_n}(x_n). \tag{4.4.2}$$

4.4.1 Leaf node message

Leaf nodes, as defined in Section 3.2, are nodes of degree $\mathcal{D} = 1$, i.e., they only have a single neighbour. These nodes are essential for initialising the BP algorithm, since they do not require any other node to transmit a message before them. Furthermore, the outgoing messages of these nodes will remain constant throughout, thus they do not require any incoming message until the final iteration of the algorithm.

As a result, unobserved variable leaf nodes $x_{\varnothing}$ do not relay any information, since they do not contain any information themselves. This is confirmed when we consider (4.4.2) where the product is empty for a leaf node and the message collapses to

$$\mu_{x_{\varnothing} \to f_m}(x_{\varnothing}) = \prod_{f_i = \varnothing} \mu_{f_i \to x_{\varnothing}}(x_{\varnothing}) = 1.$$

Similarly, factor leaf nodes $f_{\varnothing}(\mathbf{x}_{\varnothing})$ will only transmit the information they possess at initialisation. This is validated when we consider what happens to equation (4.4.1) when applied as a leaf node. Where $\mathbf{x}_{\varnothing} = \{x_n\}$, the product and summation is empty, thus

$$\mu_{f_{\varnothing} \to x_n}(x_n) = \sum_{x_i = \varnothing} \left(f_{\varnothing}(\mathbf{x}_{\varnothing}) \prod_{x_i = \varnothing} \mu_{x_i \to f_m}(x_i) \right) = f_{\varnothing}(x_n).$$

4.4.2 Observed variables

The observation of a variable gives us complete knowledge of the state of that variable. This effectively collapses its probability density to the observed value. This distribution is degenerate, since it only consists of a single value such that

$$P(observed) = \begin{cases} 1 & \text{for } \mathcal{V} \\ 0 & \text{otherwise} \end{cases},$$

where $\mathcal{V}$ is the observed value. This observation may also be expressed as

$$P(observed = \mathcal{V}) = 1.$$

Furthermore, any PD with zero probability values may bias the final marginals in the graph. However, the marginalisation in (4.4.1) removes this degeneration from the rest of the graph.

4.4.3 Approximated belief propagation (Min-sum)

Most often when applying Forward Error Correction (FEC) codes it is only necessary to find the most likely solution of the unknown variables given the observations. In these cases it is no longer required to find the true PD, but only the values that maximises it. This allows for a reduction in complexity by reducing the marginalisation of (4.4.1) from a summation to a maximisation, such that

$$\mu_{f_m \to x_n}(x_n) = \max_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \left(f_m(\mathbf{x}_m) \prod_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \mu_{x_i \to f_m}(x_i) \right).$$

In practice, in order to prevent numeric overflow, the max-product algorithm is most often carried out in the negative log likelihood domain. This will replace the product operations with summations and the maximisation

operation with a minimisation. Definition 4.1 now becomes:

Definition 4.2: The FG messages in the log likelihood domain [2, p. 339] [41, 43]

Updating rule for factor to variable:

$$\mu_{f_m \to x_n}(x_n) = \min_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \left(f_m(\mathbf{x}_m) + \sum_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \mu_{x_i \to f_m}(x_i) \right). \tag{4.4.3}$$

Updating rule for variable to factor:

$$\mu_{x_n \to f_m}(x_n) = \sum_{\substack{f_i = \\ \mathbf{f}_n \setminus f_m}} \mu_{f_i \to x_n}(x_n). \tag{4.4.4}$$

This form of the belief propagation algorithm is known as the Viterbi or Min-sum algorithm.

4.5 Belief propagation execution schedules

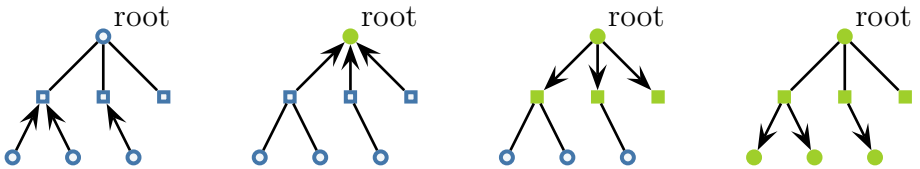
There exist many possible execution schedules by which to implement the sum-product algorithm on FGs. However, in this section we only consider two of the more commonly used schedules, and refer the reader to [41] for other message-passing schedules.

4.5.1 Standard forward-backward execution

The sum-product algorithm is the most effective when applied to FGs that are treelike. This is due to the fact that the algorithm may be terminated with an exact solution and treelike graphs are guaranteed to have leaf nodes, which provide convenient starting points.

Consider the example given in Fig. 4.5. For the sake of convenience, any variable may be chosen as the root and the graph topographically rearranged so that the graph cascades downwards from the chosen root node. The messages are now transmitted upwards, starting from the bottom level as in Fig. 4.5(a). Once the messages are transmitted, the recipient nodes will now have enough

information to transmit their messages to their remaining neighbours. Continuing this process, the messages will move up the graph until the root has received a message form all its neighbours (Fig. 4.5(b)).



(a) 1^{st} pass (b) 2^{nd} pass (c) 3^{rd} pass (d) final pass

Figure 4.5: The figure depicts the order in which the sum-product algorithm transmits its messages when the standard forward/backward execution sequence is applied. In each sub-figure the arrows indicate a message transmitted. The green nodes have received a message from all their neighbours and thus their exact marginal may be calculated.

At this stage, if we are only interested in the marginal of the root node, the algorithm is complete. However, by transmitting all of the root node's messages back downwards (Fig. 4.5(c)) and continuing all the way down to the bottom, all possible messages would have been transmitted (Fig. 4.5(d)). This will enable us to calculate all the single variable marginal PDs as well as the joint PD of all of the variables. This is commonly referred to as the forward-backward BP method.

4.5.2 Message flooding (synchronous BP)

Unfortunately, the forward-backward method reaches a deadlock if one or more cycles exist in the graph. In other words, at some stage in the algorithm there will exist no node that has received a message from all but one of its neighbours. The message flooding method resolves this issue by transmitting a message from all similar types of nodes at once regardless of how many messages each node has received, as depicted in Fig. 4.6.

For FGs, the method is initialised by assuming that all variable nodes are leaf nodes, which is known as *lazy initialisation*. Thus initially

$$\mu_{x_m \to f_w}(x_m) = 1 \quad \text{for all } m \text{ and } w.$$

Subsequently, all factor nodes will receive a message across all of their edges at once, enabling them in tun to transmit a new message back to those same vari-

(a) (b)

Figure 4.6: The figure depicts the order in which the BP transmits its messages when applied with the message flooding sequence. In each sub-figure the arrows indicate a message transmitted. The algorithm will switch between (a) and (b) until all the PDs converge.

able nodes. The result is an iterative process, alternating between equations (4.4.1) and (4.4.2). These messages are propagating incomplete information across the graph. However, with enough iterations the information of each node will propagate through the entire graph.

For treelike graphs, the algorithm is guaranteed to converge to an exact solution after a number of iterations equal to the maximum number of edges between any two nodes. However, for graphs containing cycles this process of alternating between the two updating rules does not necessarily converge and it will also not give us the exact solution. Despite this shortfall, this method remains of practical value since each message can be calculated simultaneously, making it ideal for parallelisation.

4.5.3 Sequential Message-passing (asynchronous BP)

It turns out that assigning an order to the sequence of messages for a loopy graph increases the success rate of the BP algorithm [1]. This is known as sequential message-passing and implies that only one message is passed at any point in time.

A common approach to sequential message-passing is to repeatedly apply the standard forward-backward execution to a selection of sub-graphs from the existing graph. These sub-graphs are selected such that each sub-graph is treelike. Furthermore, all clusters in the original graph must be included in at least one of the sub-graphs [39, p. 568].

After the sub-graphs have been selected, the standard forward-backward execution is applied to each of the sub-graphs for each iteration. The algorithm is halted once convergence is achieved or a maximum number of iterations is reached.

The main difference between synchronous and asynchronous BP is that where the former uses a constant set of data for each iteration, the latter always uses the most up-to-date information. This method converges to a solution more often than message flooding and, on average, with less iteration.

4.6 Tanh-rule for belief propagation

From Definitions 4.1 and 4.2 we find that the messages over FGs are confined to only a single variable, although this is not true for all implementations of the algorithm, as in the case of CGs and JTs. However, if we apply this as a constraint and assume that all variables are BRVs, we may considerably reduce the complexity of the BP algorithm [41].

We begin by considering the case where we have 2 independent BRVs, x_0 and x_1 with respective discrete probabilities $P(x_0=0)=p_0^{(x_0)}$, $P(x_0=1)=p_1^{(x_0)}$, $P(x_1=0)=p_0^{(x_1)}$, and $P(x_1=1)=p_1^{(x_1)}$. By enforcing even parity we have

$$\begin{aligned}
P(x\oplus y=0) &= p_0^{(x_0)}p_0^{(x_1)}+p_1^{(x_0)}p_1^{(x_1)}\\
&= (p_0^{(x_0)}+p_1^{(x_0)})(p_0^{(x_1)}+p_1^{(x_1)})-p_0^{(x_0)}p_1^{(x_1)}-p_1^{(x_0)}p_0^{(x_1)}\\
&= (p_0^{(x_0)}+p_1^{(x_0)})(p_0^{(x_1)}+p_1^{(x_1)})+(p_0^{(x_0)}-p_1^{(x_0)})(p_0^{(x_1)}-p_1^{(x_1)})-\\
&\quad (p_0^{(x_0)}p_0^{(x_1)}+p_1^{(x_0)}p_1^{(x_1)})\\
&= \frac{1}{2}\Big((p_0^{(x_0)}+p_1^{(x_0)})(p_0^{(x_1)}+p_1^{(x_1)})+(p_0^{(x_0)}-p_1^{(x_0)})(p_0^{(x_1)}-p_1^{(x_1)})\Big).
\end{aligned}$$

By induction we can extend this for an arbitrary finite number of BRVs x_i for $i=0,1,2\ldots,n,\ldots,N$. Then we have

$$P\left(\bigoplus_{i=0}^{N}x_i=0\right)=\frac{1}{2}\left(\prod_{i=0}^{N}(p_0^{(x_i)}+p_1^{(x_i)})+\prod_{i=0}^{N}(p_0^{(x_i)}-p_1^{(x_i)})\right) \tag{4.6.1}$$

and similarly,

$$P\left(\bigoplus_{i=0}^{N}x_i=1\right)=\frac{1}{2}\left(\prod_{i=0}^{N}(p_0^{(x_i)}+p_1^{(x_i)})-\prod_{i=0}^{N}(p_0^{(x_i)}-p_1^{(x_i)})\right). \tag{4.6.2}$$

Notice that the term $\prod_{i=0}^{N}(p_0^{(x_i)}+p_1^{(x_i)})=1$, however, we will not implement this simplification as we will use this in the identity given in (4.6.7).

The Log-Likelihood Ratio (LLR) of the BRV α is given as

$$L(\alpha)\overset{\text{def}}{=}\ln\left(\frac{P(\alpha=0)}{P(\alpha=1)}\right), \tag{4.6.3}$$

which implies the following relationships:

$$P(z=0) = \frac{e^{L(z)}}{1+e^{L(z)}} \tag{4.6.4}$$

$$P(z=1) = \frac{1}{1+e^{L(z)}}. \tag{4.6.5}$$

Furthermore, we will use the symbol $\boxplus$ as the notation for the addition defined as

$$\begin{aligned} \boxplus_{i=0}^{N} L(x_i) &\stackrel{\text{def}}{=} L(x_0) \boxplus L(x_1) \boxplus \ldots \boxplus L(x_N) \\ &\stackrel{\text{def}}{=} L(x_0 \oplus x_1 \oplus \ldots \oplus x_N). \end{aligned} \tag{4.6.6}$$

Subsequently, the following additional rules also apply

$$\begin{aligned} L(z) \boxplus \quad \infty &= \quad L(z) \\ L(z) \boxplus -\infty &= -L(z) \\ L(z) \boxplus \quad 0 \quad &= \quad 0 \end{aligned}$$

By substituting (4.6.1) and (4.6.2) into (4.6.3) we can extend equation (4.6.6) such that

$$\begin{aligned} \boxplus_{i=0}^{N} L(x_i) \equiv L\left(\bigoplus_{i=0}^{N} x_i\right) &= \ln\left(\frac{P\left(\bigoplus_{i=0}^{N} x_i = 0\right)}{P\left(\bigoplus_{i=0}^{N} x_i = 1\right)}\right) \\ &= \ln\left(\frac{\prod_{i=0}^{N}\left(e^{L(x_i)}+1\right) + \prod_{i=0}^{N}\left(e^{L(x_i)}-1\right)}{\prod_{i=0}^{N}\left(e^{L(x_i)}+1\right) - \prod_{i=0}^{N}\left(e^{L(x_i)}-1\right)}\right). \end{aligned}$$

Using the identity

$$\tanh\left(\frac{z}{2}\right) = \frac{e^z - 1}{e^z + 1} \tag{4.6.7}$$

we get

$$\boxplus_{i=0}^{N} L(x_i) \equiv \ln\left(\frac{1+\prod_{i=0}^{N}\frac{e^{L(x_i)}-1}{e^{L(x_i)}+1}}{1-\prod_{i=0}^{N}\frac{e^{L(x_i)}-1}{e^{L(x_i)}+1}}\right) = \ln\left(\frac{1+\prod_{i=0}^{N}\tanh\left(\frac{L(x_i)}{2}\right)}{1-\prod_{i=0}^{N}\tanh\left(\frac{L(x_i)}{2}\right)}\right).$$

For the sake of simplification, let us define $\kappa \equiv \prod_{i=0}^{N} \tanh\left(\frac{L(x_i)}{2}\right)$. If we then apply the above identity again, we have

$$\begin{aligned}\tanh\left(\frac{1}{2}\boxplus_{i=0}^{N} L(x_i)\right) &= \frac{e^{\ln\left(\frac{1+\kappa}{1-\kappa}\right)} - 1}{e^{\ln\left(\frac{1+\kappa}{1-\kappa}\right)} + 1} \\ &= \frac{(1+\kappa)-(1-\kappa)}{(1+\kappa)+(1-\kappa)} \\ &= \kappa.\end{aligned}$$

Therefore we conclude that

$$\boxplus_{i=0}^{N} L(x_i) \equiv 2\tanh^{-1}\left(\prod_{i=0}^{N}\tanh\left(\frac{L(x_i)}{2}\right)\right).$$

It is easy to apply this to the updating rules of Definition 4.1, where the variable to factor message in (4.4.2) is simply converted to the LLR domain as in the Viterbi algorithm:

$$\begin{aligned}L\big(\mu_{x_m\to f_w}(x_m)\big) &= L\left(\prod_{\substack{f_i=\\ \mathbf{f}_m\setminus f_w}} \mu_{f_i\to x_m}(x_m)\right) \\ \lambda_{x_m\to f_w}(x_m) &= \sum_{\substack{f_i=\\ \mathbf{f}_m\setminus f_w}} \lambda_{f_i\to x_m}(x_m). \qquad (4.6.8)\end{aligned}$$

However, the LLR transformation of the updating rule for factor to variable is somewhat more elaborate. Note that it is the factor's distribution $f_m(\mathbf{x}_m)$ in (4.4.1) that enforces the even parity and thus the summation over this distribution can be represented as a modulo-2 addition such that

$$\sum_{\substack{x_i=\\ \mathbf{x}_w\setminus x_m}} \big(f_w(\mathbf{x}_w)P(\mathbf{x}_w)\big) \equiv \bigoplus_{\substack{x_i=\\ \mathbf{x}_w\setminus x_m}} P(\mathbf{x}_w)$$

With this in mind we have:

$$
\begin{aligned}
L\left(\mu_{f_w \to x_m}(x_m)\right) &= L\left(\sum_{\substack{x_i = \\ \mathbf{x}_w \setminus x_m}} \left(f_w(\mathbf{x}_w) \prod_{\substack{x_i = \\ \mathbf{x}_w \setminus x_m}} \mu_{x_i \to f_w}(x_i)\right)\right) \\
\lambda_{f_w \to x_m}(x_m) &= L\left(\bigoplus_{\substack{x_i = \\ \mathbf{x}_w \setminus x_m}} \prod_{\substack{x_i = \\ \mathbf{x}_w \setminus x_m}} \mu_{x_i \to f_w}(x_i)\right) \\
&= \boxplus_{\substack{x_i = \\ \mathbf{x}_w \setminus x_m}} \lambda_{x_i \to f_w}(x_i) \\
&= 2 \tanh^{-1}\left(\prod_{\substack{x_i = \\ \mathbf{x}_w \setminus x_m}} \tanh\left(\frac{\lambda_{x_i \to f_w}(x_i)}{2}\right)\right). \qquad (4.6.9)
\end{aligned}
$$

The advantage of the final update rules is that all operations are applied to only a single value, which implies that the tanh-rule algorithm's complexity scales with $O(N)$, where N is the number of variables in the graph. This is a significant reduction in complexity if we consider that the complexity of the update rules of the Viterbi and sum-product algorithms scale with $O(2^N)$.

4.7 Belief propagation over cluster graphs

The application of the sum-product algorithm over CGs may be considered to be a more generalised form of BP, which is supported by the fact that all other graphs covered in this thesis are special cases of CG.

Since multiple variables may be shared between two neighbouring clusters, for a tree-like graph the messages passed between them are defined over the intersection of their scopes. That is, the message between clusters $\phi_0(\mathbf{x}_0)$ and $\phi_1(\mathbf{x}_1)$ has the scope $\mathbf{x}_0 \cap \mathbf{x}_1$.

For CGs graph with loops, the Running Intersection Property (RIP) must be taken into consideration. Consequently, the scope of the messages is equal to that of the scope of the sepset of the respective clusters, which is a subset of the intersection of their scopes. For example, should clusters $\phi_0(\mathbf{x}_0)$ and $\phi_1(\mathbf{x}_1)$ share the sepset $\psi_S(\mathbf{x}_S)$, we have $\mathbf{x}_S \subseteq (\mathbf{x}_0 \cap \mathbf{x}_1)$.

Beyond this, the principles remain the same as before. For example, in Fig. 4.7, the cluster $\phi_0(\mathbf{x}_0)$ starts by collecting the messages of the clusters preceding it, i.e., $\mu_{\phi_2 \to \phi_0}(\mathbf{x}_{S1})$ and $\mu_{\phi_3 \to \phi_0}(\mathbf{x}_{S2})$. By combining these with its

own distribution and marginalising over all variables not in set $\mathbf{x}_1$, the message transmitted to cluster $\phi_1(\mathbf{x}_1)$ is obtained. The general update rule is given as:

Definition 4.3: The update rule of BP over CGs [39, p. 113]

$$\mu_{\phi_m \to \phi_n}(\mathbf{x}_S) = \sum_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \left(\phi_m(\mathbf{x}_m) \prod_{\substack{\phi_i = \\ \phi_m \setminus \phi_n}} \mu_{\phi_i \to \phi_m}(\mathbf{x}_T) \right), \tag{4.7.1}$$

where $\mathbf{x}_S \subseteq (\mathbf{x}_m \cap \mathbf{x}_n)$ and $\mathbf{x}_T \subseteq (\mathbf{x}_i \cap \mathbf{x}_m)$.

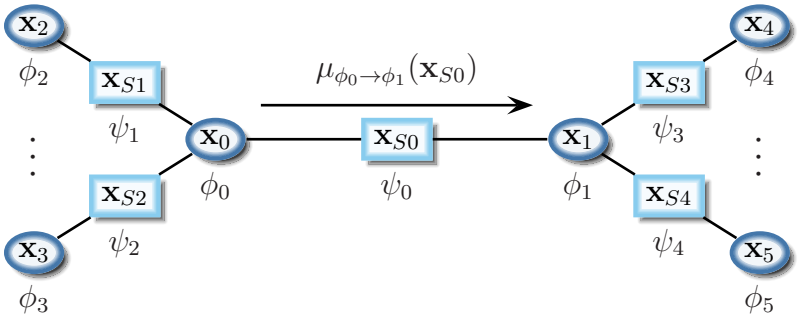


Figure 4.7: A CG with an arbitrary number of nodes. In order for $\phi_0(\mathbf{x}_0)$ to transmit $\mu_{\phi_0 \to \phi_1}(\mathbf{x}_{S0})$ to $\phi_1(\mathbf{x}_1)$, it must first receive the messages of the all clusters preceding it, i.e., $\mu_{\phi_2 \to \phi_0}(\mathbf{x}_{S1})$ and $\mu_{\phi_3 \to \phi_0}(\mathbf{x}_{S2})$, etc.This message is created by use of the general update rule as given in Definition 4.3.

The advantage of this more generalised form of BP lies in the fact that these messages' scopes are not restricted to a single variable. This allows for the transmission not only of the PD of the respective variables, as *derived* from its relationship with the other variables, but of the dependencies between them as well. That is, more information is being propagated for every message passed [1, p. 406].

As a result we should find that, on average, a CG will converge in fewer iterations than its equivalent FG. Moreover, we should find that the CG converges more often than the FG, as state by Koller in [1, p. 428]. Unfortunately, this does not guarantee a reduction in complexity for CGs compared to FGs, since the tanh-rule covered in Section 4.6 significantly reduces the complexity of BP and is not applicable to multi-variable messages.

4.8 Summary

In Chapter 3 we found that PGMs serve as a useful tool to depict the relationship and inter-dependencies of a set of variables. In this chapter their true potential comes to fruition once inference is applied to them.

We considered the message-passing principles that most of these probabilistic models are based upon in Section 4.2. This scheme works for any network that has a treelike-graph structure and an order of complexity proportional to the number of nodes. It is started at the *leaf nodes* and terminates once all nodes have transmitted a message to all of its neighbours, at which stage all of the nodes contain all the information initially within the network. Also, each node must only receive the information exactly once, otherwise the information will be duplicated and the final result skewed. Unfortunately, in the case of graphs with loops, this is not possible to guarantee and thus it is impossible to guarantee an exact solution as the end result.

In Section 4.3 we covered the use of the VE algorithm to reduce the complexity of marginal inference by dividing the process into multiple steps. We find that the VE algorithm's complexity scales in a manner that is linearly proportional to the number of variables, in contrast with the exponential complexity scaling of the naive approach. Furthermore, for each elimination, the VE algorithm changes the structure as well as the potentials of the MRF of the given PD, as shown in Fig. 4.3. The update of the remaining potentials can be considered as the passing of a message to the neighbouring node(s).

Section 4.4 gives a qualitative description of how BP extends the concept of message-passing in order to apply inference over PGMs. Also known as the sum-product algorithm, it involves only the two operations from which the name is derived, i.e., summations and products. Since there are two types of nodes in a FG, the sum-product involves two types of messages for these types of graphs as shown in Fig. 4.4 and defined in Definition 4.1. Unobserved variable leaf nodes do not relay any information, since they do not contain any information themselves. Similarly, factor leaf nodes will only transmit the information they initially possessed.

In some practical cases it is not required to find the true PD, but only the values that maximise it. This allows for a reduction in complexity by reducing the marginalisation of the PD to a maximisation of the PD, and operating in the LLR domain. This form of the belief propagation algorithm is known as

the Viterbi or Min-sum algorithm.

In Section 4.5 we only consider three of the more efficient and widely used schedules, namely:

- Standard forward-backward execution
- Message flooding (synchronous BP)
- Sequential message-passing (asynchronous BP).

For treelike graphs, the algorithms are guaranteed to converge to an exact solution. This is not the case for graphs containing cycles, and yet it remains of practical importance as it often does converge to an approximate solution.

The *Tanh-rule*, in Section 4.6, reduces the complexity of the BP from $O(2^{\mathcal{N}})$ to $O(\mathcal{N})$, where $\mathcal{N}$ is the number of variables in the graph. However, this approach requires that the messages be confined to a single BRV, which severely limits its usefulness.

BP over CGs may be considered to be a more generalised form of BP, as explained in Section 4.7. The advantage of this more generalised form of BP lies in the fact that these messages' scopes are not restricted to a single variable. This allows for more information to be propagated for every message passed. Thus, on average, a CG will converge in fewer iterations than its equivalent FG. Moreover, fewer messages implies that less potential biasing will occur in the case of graphs with cycles.

Chapter 5

Applying PGMs to Raptor codes

5.1 Introduction

The potential applications of Probabilistic Graphical Models (PGMs) are vast. Even limited to Forward Error Correction (FEC), the number of possible approaches is extensive. In this chapter we will limit our investigation to the PGMs models covered in Chapter 3, that is, the Factor Graph (FG), Cluster graph (CG), and the Junction Tree (JT), and apply them to the Raptor code. We focus on the use of Belief Propagation (BP) techniques to decode the original input symbol values for the general Raptor code.

In Section 5.2 we cover how PGMs can describe the transmission of symbols over a given channel. We also see how these descriptions may be incorporated so that no unnecessary messages are passed during the iterative decoding process of the BP algorithm. Following this, the compilation of the standardised Raptor 10 (R10) code is explained in Section 5.3. This standard Raptor code is used as a real-world application upon which the comparative simulations in the next chapter are based.

Thereafter, Section 5.4 covers the process of applying the FG model to a given Raptor code. In Section 5.5, two geometric alteration techniques are explained that may show improved performance when applying BP. Finally, the applications of CGs (in Section 5.6) and JTs (in Section 5.7) to the Raptor code are given, as well as how they are decoded.

5.2 Channel coding with PGMs

In channel coding, the PGM of an FEC code may be applied from 2 perspectives: that of the transmitter or that of the receiver. From the latter viewpoint the PGM will deduce the likelihood that said information was received correctly. From the former viewpoint, it will infer the probability that the data was transmitted successfully. For example, for transmission over the Binary Erasure Channel (BEC), the PGM will infer the probability that a transmitted symbol was not lost due to an erasure. Similarly, for transmission over the Binary Symmetric Channel (BSC), the PGM will infer the probability that a bit-flip do not occur.

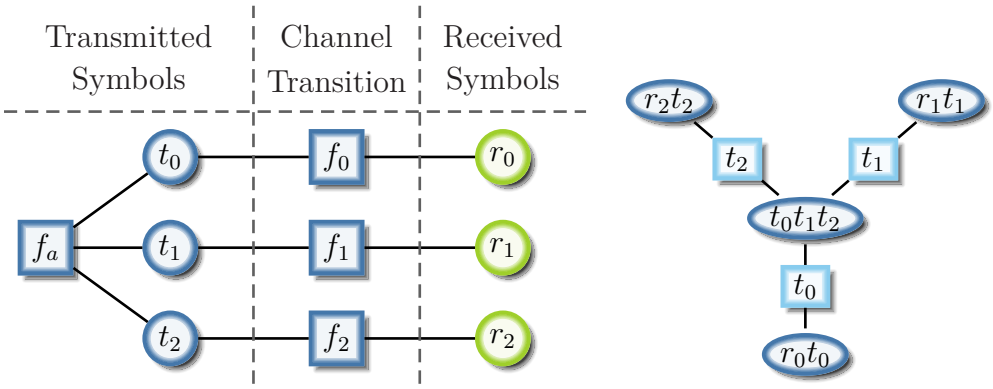
Since we are focused on the decoding of the FEC code, we will consider the graph from the receiver end, unless specified otherwise. Despite the distinction, in both cases the underlying theory and application thereof is the same, even though the population of the Probability Distributions (PDs) in the graph may differ depending on the viewpoint taken.

The assembly of an FEC code's data happens at the transmitter. Thus all initial symbols in the PGM of any given FEC code are unobserved by the receiver. The received symbols are the only *observed* values, aside from the structure of the graph itself which is assumed to be known to the receiver, e.g., via package headers or a predefined synchronous protocol [8].

The observed values do not pertain to any of the variables included in the original graph as constructed by the transmitter and must therefore be added as additional entities. As depicted in Fig. 5.1, in FGs each observation is related to the original graph by the addition of a new node. Each new node is connected to its corresponding transmitted variable by means of a factor containing the transition PD of the channel involved at the moment of transition. For all *memoryless* channels[1] these transition PDs will be identical. Again, the topographical relationship between the observed and unobserved symbols is assumed to be known to the receiver due to the protocol at hand.

Incorporating the observed values into a CG involves a similar process where a new node is added with a scope that includes both the observed variable and *all* the unobserved symbols associated with it. The new node must be connected to the existing graph, such that both the Running Inter-

[1] A channel is described as "memoryless" if the input and output at time $T = t$ are independent of all other inputs and outputs at time $T \neq t$ [3]



(a) Factor Graph (b) Cluster graph

Figure 5.1: A minimalistic example of both the FG and CG representation of the received symbols r_i that are related to their respective transmitted symbols t_i for $i = 0, 1, 2$. The FGs nodes are grouped according to what their PDs describe.

section Property (RIP) and the family preservation constraints are maintained. This implies that the resulting sepset includes all but the observed variable included in the new node's scope.

Even though both the graphs in Fig. 5.1 have a tree-like topography, this is almost never the case for real-world applications of Raptor codes. However, what is guaranteed to be tree-like is the newly added subgraphs of the observed symbols due to the one-to-one relationship they share with their counterparts. This is the case for both FGs and CGs. Since all the observed symbols are also leaf nodes, it is possible for absolute inference to be applied to these subgraphs. Doing so reduces the computational complexity of the graph before running the iterative process of loopy BP.

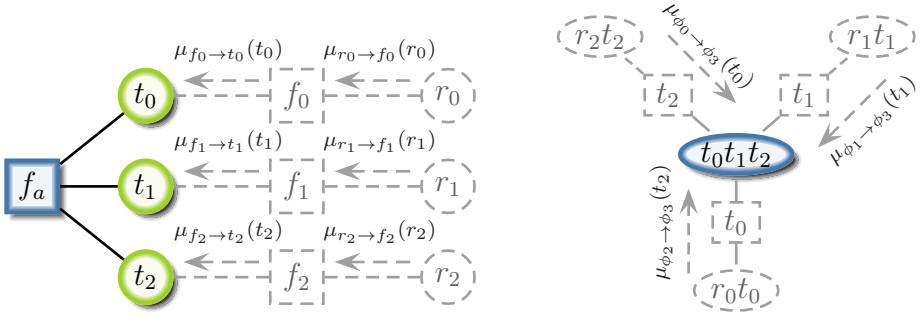
In other words, the marginal PD of the unobserved symbols can be updated wherever they have a directly associated observed variable. For example, from Fig. 5.1 we have

$$\begin{aligned} P(t_0) &\propto \mu_{f_a \to t_0}(t_0) \sum_{r_0} \Big(f_0(t_0, r_0) \mu_{r_0 \to f_0}(r_0) \Big) \\ &= \mu_{f_a \to t_0}(t_0) \sum_{r_0} \Big(f_0(t_0, r_0) P(t_0 | r_0 = \mathcal{V}) \Big) \\ &= \mu_{f_a \to t_0}(t_0) P_{\mathcal{V}}(t_0), \end{aligned}$$

where $\mathcal{V}$ is the observed value and $P_{\mathcal{V}}(t_0)$ is the resulting PD received by node t_0 from node f_0. Subsequently, we have

$$\mu_{t_0 \to f_a}(t_0) = P_{\mathcal{V}}(t_0).$$

Since the received symbol to transmitted symbol subgraphs are tree-like, their messages will remain constant throughout the process of the BP algorithm. Thus, the PD of the neighbour of all the received symbol nodes may be updated and the received symbol itself removed, as is done in the Variable Elimination (VE) algorithm. The reduced form of the graphs given in Fig. 5.1 is shown in Fig. 5.2.



(a) Factor Graph (b) Cluster graph

Figure 5.2: The FG and CG examples in Fig. 5.1 with reduced complexity. Here VE is applied to the nodes relating to the received symbols. That is, their nodes are removed and their messages are passed on to the transmitted symbol nodes. Any BP applied to these graphs will result in the same answers as that of their more complex counterparts in Fig. 5.1.

Thereafter, loopy BP is applied to the constructed graph in order to obtain the local maxima of the transmitted data, i.e., their most probable values.

5.3 The R10 standardised Raptor Code

Optimisation of both the precode and the Luby Transform (LT) code sections of a Raptor code is a formidable task. Considerable work has been done in this regard, with a number of different approaches. Although most of this work is focused on the optimisation of the Raptor code for the BEC, designing the Raptor code for BSC and Binary Additive White Gaussian Noise Channel (BAWGNC) is beyond the scope of this thesis. Moreover, it is shown in [17] that Raptor codes designed for the BEC perform well for BSC and BAWGNC, even though it is postulated therein that some improvement is possible with an optimised design.

In light of this, the R10 code was chosen for use in our measurements. It is the first standardised Raptor code that has been adopted into a number of different standards. These include 3rd Generation Partnership (3GPP) Multimedia Broadcast/Multicast Service (MBMS) [13], Internet Engineering Task Force (IETF) IETF [14] and many others [8]. The R10 code is covered in great length in the work of M.A. Shokrollahi and M. Luby [8], which is our main source regarding the R10 code. Even though the RaptorQ (RQ) code [8, 44] has made some significant improvements, it is not directly relevant to the focus of this work. Accordingly, the R10 was chosen for its lower complexity compared to the RQ code.

The R10 is a systematic code designed to encode up to 2^{13} source symbols and 2^{16} output symbols. It is designed specifically for the application of the Inactivation Decoding (ID) without the use of graphs. We will cover an adaptation of the ID algorithm for use with graphs in Section 5.5.2. For now, suffice it to say that ID is based on the well known Gaussian Elimination (GE) algorithm and is applied to the generation matrix of the LT section of the R10 code.

GE is vulnerable to rank deficiency in the matrix it is applied to. That is, the matrix GE is applied to must have at least $\mathcal{K}$ independent rows in order to decode $\mathcal{K}$ unique symbols. Subsequently, ID shares the same vulnerability. For this reason, the R10's precode is designed to maximise the likelihood that the generation matrix $\mathbf{G}$ is generated with full rank, since only if $\mathbf{G}$ is full rank can the given number of output symbols be decoded. $\mathbf{G}$ is a combination of the precode and LT code generation matrices. That is, given the precode generation matrix $\mathbf{G}_y$ and LT-code generation matrix $\mathbf{G}_t$, we have

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_t \\ \mathbf{G}_y \end{bmatrix}$$

This is achieved by appending a parity-check matrix of an Low-Density Parity-Check (LDPC) code with that of High-Density Parity-Check (HDPC) code that behaves as if its rank properties were uniformly random, without being uniformly random itself. For the R10 code, a binary reflected gray code $\hat{\mathbf{B}}$ [45] is used to construct such an HDPC sub-matrix.

A binary reflected gray code is a sequence of binary numbers

$$\boldsymbol{\tau} = \begin{bmatrix} \tau_0 & \tau_1 & \dots & \tau_{M-1} \end{bmatrix}$$

for $M \leq 2^N$, each having N number of bits. The numbers are defined so that τ_m and τ_{m+1} for $m = 0, 1, \ldots, M$ only differ by a single bit. For example, one possible binary reflected gray code with $N = 3$ and $M = 8$ is

$$\boldsymbol{\tau} = \begin{bmatrix} 000 & 001 & 011 & 010 & 110 & 100 & 101 & 111 \end{bmatrix} \tag{5.3.1}$$

For the R10 code, the bits of the numbers in the binary reflected grey code are rearranged in the matrix $\hat{\mathbf{B}}$. This is done such that column i represents the bits in the i^{th} binary number. For example, the binary reflected grey code in (5.3.1) will become

$$\hat{\mathbf{B}} = \begin{array}{c} \begin{matrix} \tau_0 & \tau_1 & \tau_2 & \tau_3 & \tau_4 & \tau_5 & \tau_6 & \tau_7 \end{matrix} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \end{array}.$$

However, the R10 code restricts the number of 1s in each column in $\hat{\mathbf{B}}$, referred to as the column weight, to equal half of the column length, rounded to the nearest integer. This implies that each column in $\hat{\mathbf{B}}$ will alter with 2 bits, instead of only one. The Hamming distance between consecutive columns is thus 2.

Given $\mathcal{K}$ source symbols, the R10 code will produce $\mathcal{L}$ number of LDPC parity symbols and $\mathcal{H}$ number of HDPC parity symbols. The total number of input symbols amounts to $\mathcal{N} = \mathcal{K} + \mathcal{L} + \mathcal{H}$. The layout of the R10 precode parity-check matrix $\mathbf{H}$ is defined as a compilation of several sub-matrices such that:

$$\mathbf{H} = \left[\begin{array}{ccccc:c} \mathbf{C}_0 & \mathbf{C}_1 & \mathbf{C}_{\ldots} & \mathbf{C}_i & \mathbf{I}_{\mathcal{L}} & \mathbf{0} \\ \multicolumn{5}{c:}{\hat{\mathbf{B}}} & \mathbf{I}_{\mathcal{H}} \end{array}\right]. \tag{5.3.2}$$

The sub-matrix $\begin{bmatrix} \mathbf{C}_0 & \mathbf{C}_1 & \mathbf{C}_{\ldots} & \mathbf{C}_i & \mathbf{I}_{\mathcal{L}} \end{bmatrix}$ is the LDPC section of the R10 precode, where $\mathbf{C}_i$ is the i^{th} circulant matrix and $\mathbf{I}_{\mathcal{L}}$ is an identity matrix. Each of these matrices has a size of $\mathcal{L} \times \mathcal{L}$, except the last circulant matrix, which is $\mathcal{L} \times (\mathcal{K} mod \mathcal{L})$. The LDPC portion is constructed to ensure that the input symbols are encoded approximately the same number of times.

Each circulant matrix has columns of weight 3, where the positions of the 1s in the first column for the i^{th} matrix are given as

$$0, \ (i+1) mod(\mathcal{L}), \ \text{and} \ (2i+1) mod(\mathcal{L}).$$

The number of LDPC symbols is the smallest prime number that satisfies the inequality

$$\mathcal{L} \geq \lceil 0.01\mathcal{K} \rceil + X \text{ where } \lfloor X \rfloor (\lfloor X \rfloor - 1) \geq 2\mathcal{K}$$

The sub-matrix $\mathbf{0}$ is a zero matrix of size $\mathcal{L} \times \mathcal{L}$.

The appended HDPC sub-matrix $\begin{bmatrix} \hat{\mathbf{B}} & \mathbf{I}_{\mathcal{H}} \end{bmatrix}$ consists of the $(\mathcal{K} + \mathcal{L}) \times \mathcal{H}$ grey code $\hat{\mathbf{B}}$ and the $\mathcal{H} \times \mathcal{H}$ identity matrix $\mathbf{I}_{\mathcal{H}}$. The number of HDPC symbols is the smallest integer satisfying

$$\binom{\mathcal{H}}{\mathcal{H}/2} \geq \mathcal{K} + \mathcal{L}.$$

This inequality is used to ensure that $\hat{\mathbf{B}}$ has the minimum number of rows such that each column is unique.

An example of the parity-check matrix for the R10 precode with $\mathcal{K} = 6$ is given in Fig. 5.3.

$$\mathbf{H} = \left[\begin{array}{cccccc:c:cccccc:cccccc}
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
\hdashline
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & &
\end{array}\right]$$

$$\begin{array}{cccccc:c:cccccc:cccccc}
1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hdashline
1 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{array}$$

$$\begin{array}{cccccc:c:cccccc:cccccc}
\multicolumn{6}{c}{\mathbf{C_0}} & & \multicolumn{6}{c}{\mathbf{C_1}} & & \multicolumn{6}{c}{\mathbf{I}_{\mathcal{L}}} & & \multicolumn{6}{c}{\mathbf{0}} \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
\hdashline
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & & & & & & & & & & & & \\
 & & & & & & \multicolumn{6}{c}{\hat{\mathbf{B}}} & & & & & & & & \multicolumn{6}{c}{\mathbf{I}_{\mathcal{H}}}
\end{array}$$

Figure 5.3: The precode parity-check matrix $\mathbf{H}$ of the R10 code where $\mathcal{K} = 6$, $\mathcal{L} = 5$, and $\mathcal{H} = 6$. It consists of two circular sub-matrices C_0 and C_1 with coulomb weight 3, a binary reflected gray code sub-matrix $\hat{\mathbf{B}}$, two identity sub-matrices $\mathbf{I}_{\mathcal{L}}$ of size $\mathcal{L} \times \mathcal{L}$ and $\mathbf{I}_{\mathcal{H}}$ of size $\mathcal{H} \times \mathcal{H}$ and a zero sub-matrix $\mathbf{0}$.

The number of source symbols results in the following properties of the matrix:

$$\lfloor X \rfloor (\lfloor X \rfloor - 1) \geq 2\mathcal{K} = 12$$
$$\therefore X = 4,$$

$$\begin{aligned}\mathcal{L} &\geq \lceil 0.01\mathcal{K} \rceil + X \\ \mathcal{L} &\geq 1 + 4 \\ \therefore \mathcal{L} &= 5,\end{aligned}$$

and

$$\begin{aligned}\begin{pmatrix} \mathcal{H} \\ \mathcal{H}/2 \end{pmatrix} &\geq \mathcal{K} + \mathcal{L} = 11 \\ \therefore \mathcal{H} &= 6.\end{aligned}$$

The LT section of the R10 code is generated using the degree distribution in Table 5.1. The design of the degree distribution is done using a form of asymptotic linear programming known as density evolution [9] and is optimised for use with ID.

Table 5.1: The degree distribution of the R10 code, with an average degree of 4.63. This is designed to work well with ID.

$\Omega(\mathcal{D}_o)$	0.0098	0.4590	0.2110	0.1134	0.1113	0.0799	0.0156
$\mathcal{D}_o$	1	2	3	4	10	11	40

5.4 Decoding using factor graphs

Fig. 5.4 depicts the FG of an R10 code as 3 top-to-bottom stages. The first stage, labelled *pre-factors* and *input symbols*, is the precode with the input symbol set $\mathbf{y} = \begin{bmatrix} y_0 & y_1 & \dots & y_6 \end{bmatrix}$ gained after the initial encoding of the given data set $\mathbf{z} = \begin{bmatrix} y_0 & y_1 & y_2 & y_3 \end{bmatrix}$. This is a Hamming precode, where its factors define the eXclusive-OR (XOR) relationship of the input symbols. It is a simplified form of the standard R10 precode, i.e., it has no HDPC section in its parity-check matrix $\mathbf{H}$, such that

$$\mathbf{H} = \begin{bmatrix} \mathbf{C}_3 & \mathbf{I}_4 \end{bmatrix}.$$

where $\mathbf{C}_3$ is a circulant matrix with column weights of three, that is there are three 1s in each column. $\mathbf{I}_4$ is an identity matrix of size 4×4. This simplified version is only used for illustrative purposes.

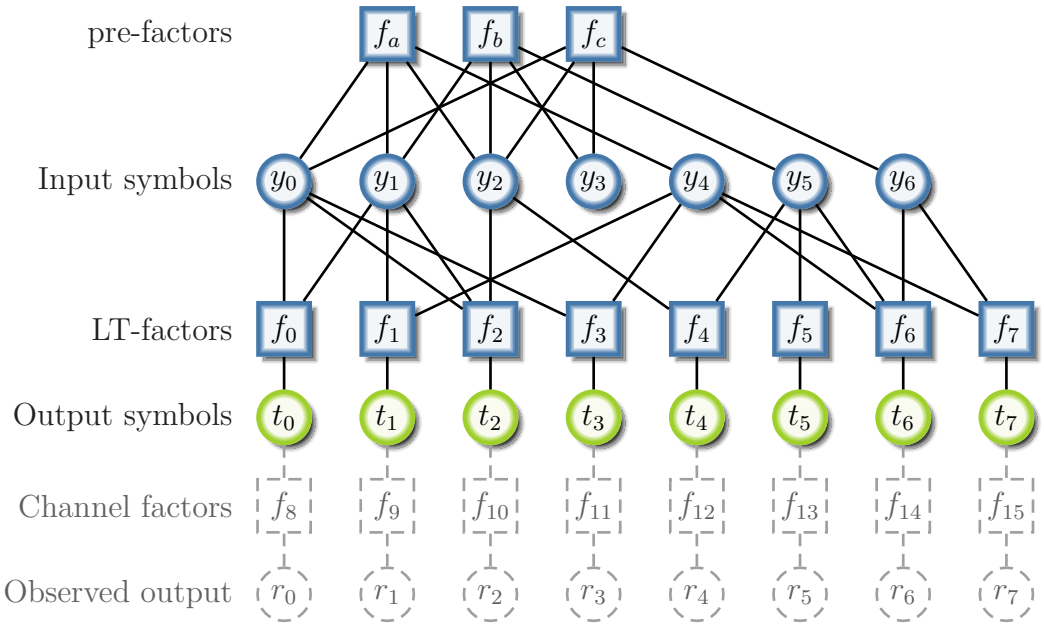


Figure 5.4: An example of a small scale Raptor code FG. It shows the typical structure of a standard R10 code, arranged so that the different sections of the code form the rows. The 2 uppermost rows depict the precode, followed by the LT section and finally the transmission section of the code over a given channel. The initial propagation of the observed variable's PD is shown to be applied.

The second stage of the graph is the LT code section consisting of the input symbols, LT-factors, and output symbols. Their relationships are defined by the generation matrix

$$\mathbf{G}_t = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \end{bmatrix}.$$

The final stage of the graph describes the transition of the symbols over the given channel. This includes the output symbols, channel factors, and the observations made by the receiver. The figure shows the initial propagation of the observed variable's PD as already applied.

Even with such a small scale example, the graph is evidently not suited for the standard forward-backward execution of the BP algorithm, hence it is

decoded using the sequential message-passing approach.

Decoding the FG is done as follows:

1. Initialise all symbol nodes' messages such that

$$\lambda_{y_n \to f_m}(y_n) = 0,$$
$$\lambda_{t_n \to f_m}(t_n) = L(t_n | r_n = \mathcal{V})$$

for all n and m, where $L(\mu) = \ln\left(\frac{P(\mu=0)}{P(\mu=1)}\right)$.

2. Update all factor-to-variable messages $\lambda_{f_m \to y_n}(y_n)$ and $\lambda_{f_m \to t_n}(t_n)$ using (4.6.9):

$$\lambda_{f_m \to y_n}(y_n) = 2 \tanh^{-1}\left(\prod_{\substack{x_i = \\ \mathbf{y}_m \setminus y_n}} \tanh\left(\frac{\lambda_{y_i \to f_m}(y_i)}{2}\right)\right).$$

3. Update all variable-to-factor messages $\lambda_{y_n \to f_m}(y_n)$ and $\lambda_{t_n \to f_m}(t_n)$ using (4.6.8):

$$\lambda_{y_n \to f_m}(y_n) = \sum_{\substack{f_i = \\ \mathbf{f}_n \setminus f_m}} \lambda_{f_i \to y_n}(y_n).$$

4. Calculate the marginal PDs of the input symbols, i.e.,

$$P(y_n) = \sum_{f_i = \mathbf{f}_n} \lambda_{f_i \to y_n}(y_n)$$

for all n.

5. Repeat Item 2 to Item 4 until

$$\frac{1}{\mathcal{N}} \sum_{y_n = \mathbf{y}} \left(L(y_n)_i - L(y_n)_{i-1}\right) < \mathcal{Q}.$$

or

$$i \geq \mathcal{I}_{max}.$$

That is, choose some threshold value $\mathcal{Q}$ such that convergence is achieved once the difference in the average of the Log-Likelihood Ratio (LLR) of all input symbol marginal PDs, from BP pass $i-1$ to BP pass i, does not exceed $\mathcal{Q}$. $\mathcal{I}_{max}$ is the selected maximum number of BP iterations allowed per decoding attempt.

6. If the number of iterations i exceeds the maximum allowed number $\mathcal{I}_{max}$, wait for a new observation to be made and add its corresponding output symbol and LT-factor to the FG. Thereafter, restart the algorithm from Item 1. If $i < \mathcal{I}_{max}$, a local maximum has been reached.

5.5 Factor graph altering enhancements of the BP algorithm

The decoding problem may be defined as the distribution of the information that was derived from the observed over the FG. Before decoding, each input symbol y is undetermined and each output symbol t contains the information that was obtained from its corresponding observed symbol r. Given an input symbol set of $\mathbf{y}$, the decoding problem is solvable once $|\mathbf{y}|$ bits worth of information are collected, where the information content of a symbol is defined as

$$H(X) \stackrel{\text{def}}{=} \sum_{x \in \mathcal{A}_X} P(x) \log_{|X|} \frac{1}{P(x)} \quad \text{bits},$$

$\mathcal{A}_X$ defines the set of possible values of the variable x and $|X|$ the size of that set. It will be assumed that each symbol is only a single bit, hence $|X| = 2$.

For an undetermined symbol, $H(X) = max\Big(H(X)\Big) = 1$. Since we require at least as many bits of information as there are symbols, this implies that at least as many output symbols are required as input symbols. However, having enough information in the global sense is not the only necessity for successful inference. We also require enough localised information to successfully transfer the information in the output symbols to the input symbols. Specifically, we need enough information in each cluster due to their inherited XOR relationship.

Take, for example, the LT factor $f_0(t_0, y_0, y_1)$ in Fig. 5.4. Assuming commutation over the BEC and thus applying deterministic inference, the relationship between its connected symbols is

$$t_0 \oplus y_0 \oplus y_1 = 0.$$

Suppose it was observed that $r_0 = 1$. Given that we may have absolute confidence in any bit that is received over a BEC, since an erasure would imply not receiving the bit, we have $t_0 = r_0$ and the previous equation then

simplifies to

$$1 \oplus y_0 \oplus y_1 = 0$$
$$y_0 \oplus y_1 = 1$$

This still leaves us with 2 possible solutions. Thus, knowing only one symbol is not enough information to obtain a unique solution. Accordingly, for a cluster of N dependent symbols, we need to know the value of $N-1$ symbols to obtain a unique solution.

This problem is also true for probabilistic inference and can be shown to be the case for FGs using (4.6.9) in which it is given that

$$\lambda_{f\to x_n}(x_n) = 2\tanh^{-1}\left(\prod_{\substack{x_i=\\ \mathbf{x}\setminus x_n}} \tanh\left(\frac{\lambda_{x_i\to f}(x_i)}{2}\right)\right)$$

where $f(\mathbf{x})$ is a factor with a scope of $\mathbf{x} = \begin{bmatrix} x_0 & x_1 & \dots & x_i & \dots & x_{\mathcal{N}} \end{bmatrix}$. Assuming that x_0 is still undetermined, i.e.,

$$\lambda_{x_0\to f}(x_0) = \log\frac{0.5}{0.5} = 0,$$

the equation becomes

$$\begin{aligned}
\lambda_{f\to x_n}(x_n) &= 2\tanh^{-1}\left(\tanh\left(\frac{\lambda_{x_0\to f}(x_0)}{2}\right)\prod_{\substack{x_i=\\ \mathbf{x}\setminus(x_0,x_n)}} \tanh\left(\frac{\lambda_{x_i\to f}(x_i)}{2}\right)\right)\\
&= 2\tanh^{-1}\left(\tanh(0)\prod_{\substack{x_i=\\ \mathbf{x}\setminus(x_0,x_n)}} \tanh\left(\frac{\lambda_{x_i\to f}(x_i)}{2}\right)\right)\\
&= 2\tanh^{-1}\left(0\prod_{\substack{x_i=\\ \mathbf{x}\setminus(x_0,x_n)}} \tanh\left(\frac{\lambda_{x_i\to f}(x_i)}{2}\right)\right)\\
&= 0
\end{aligned}$$

Thus any message degenerates into an LLR of 0 if any of the messages pertaining to it is an LLR of 0.

We therefore require a graph structure that will allow the successful propagation of the information in each output symbol across the entire graph. For larger codes we may construct such a code by using a well-designed output degree distribution.

One fundamental problem with fountain codes is that enough degree $\mathcal{D}_o = 1$ output symbols are required in order to initialise the BP algorithm and successfully decode the LT code. Conventionally, this problem is overcome by using very large fountain codes [8]. However, with only a small increase in computational complexity, we can use the relationship of the input and output symbols to change the structure of the factor graph in order to obtain the necessary $\mathcal{D}_o = 1$ output symbols.

Previously, these algorithms were only considered for BEC. However, the probabilistic relationship of the variables remains intact when these algorithms are applied. Thus it is viable to adapt the algorithm for the Raptor/LT codes for probabilistic decoding.

5.5.1 Tree-structure Expectation propagation

An algorithm named Tree-structure Expectation Propagation (TEP) is proposed and analysed in [19]. It changes the graph structure of an FG to decode LDPC codes over the BEC. In [24] this algorithm was analysed for the LT codes over the BEC.

The TEP algorithm is based on the observation that the relationships of the symbols, as defined by a given LT factor, are simultaneous XOR equations. Thus for the instance in Fig. 5.4 we have

$$t_0 = y_0 \oplus y_1 \qquad t_1 = y_1 \oplus y_2 \oplus y_4$$
$$t_2 = y_0 \oplus y_1 \oplus y_2 \qquad t_3 = y_0 \oplus y_4$$

The XOR operation is commutative, thus we may also write the first equation as $y_1 = t_0 \oplus y_0$. We can now substitute this into the remaining equations to remove y_1, leaving us with

$$t_0 = y_0 \oplus y_1 \qquad t_0 \oplus t_1 = y_0 \oplus y_2 \oplus y_4$$
$$t_0 \oplus t_2 = y_0 \oplus y_0 \oplus y_2 = y_2 \qquad t_3 = y_0 \oplus y_4$$

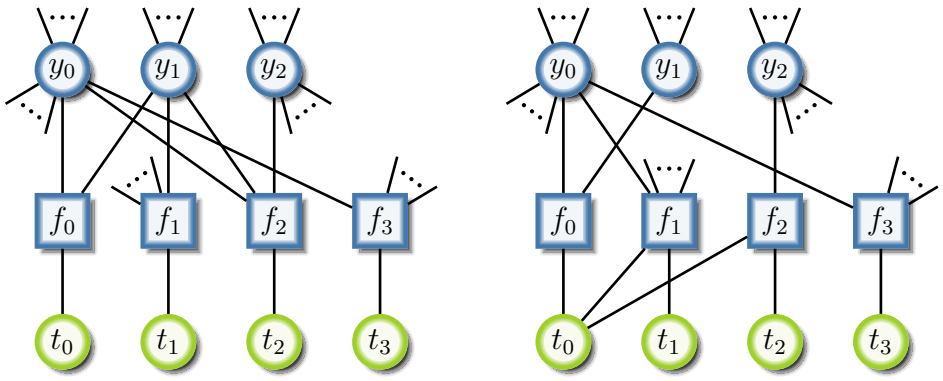
This simple sequence of operations forms the basis upon which the TEP algorithm is developed.

TEP is only executed once the BP fails due to an insufficient degree distribution. The algorithm starts by searching for the first $\mathcal{D}_o = 2$ output symbol. For an FG, such an output symbol may be identified through an LT factor

$f(x_j, x_k)$ that shares an edge with 2 input symbols, x_j and x_k. That is, the LT factor has an input symbol degree of $\mathcal{D}_i = 2$. This factor is then added to a list, which will exclude it from searches in future iterations by the algorithm.

Thereafter, one of the two input symbols x_j or x_k is randomly chosen. All LT factors other than $f(x_j, x_k)$, sharing an edge with both input symbols, have both edges removed. Each LT factor sharing an edge only with the chosen input symbol has its link switched to the other input symbol of $f(x_j, x_k)$. Furthermore, all output symbols connected to $f(x_j, x_k)$ are connected to each factor that had its links altered.

Finally, the algorithm will reinitialise the PDs of all nodes and retry the BP algorithm on the new graph. If the attempted decoding fails again, the graph alteration process will be repeated until either BP is successful or no more $\mathcal{D}_i = 2$ LT factors remain.



(a) Before TEP (b) After TEP

Figure 5.5: A sub-graph of the Raptor code in Fig. 5.4. Note that it only consists of the LT section of the code. (a) The sub-graph before a single iteration of the TEP algorithm, with $f_0(t_0, y_0, y_1)$ as the chosen $\mathcal{D}_i = 2$ LT-factor and y_1 is to be isolated. Here, $f_2(t_2, y_0, y_1, y_2)$ share both input symbols with $f_0(t_0, y_0, y_1)$, where $f_1(t_1, y_1, y_4)$ and $f_3(t_3, y_0, y_4)$ only shares one of the 2 symbols. (b) The sub-graph after 1 iteration of the TEP algorithm. Here $f_2(t_2, y_2, t_0)$ has now become a $\mathcal{D}_i = 1$ LT factor. $f_1(t_0, t_1, y_0, y_4)$'s edges have also been altered, whereas $f_3(t_3, y_0, y_4)$ remains unchanged.

For example, Fig. 5.4 will fail its initial BP attempt, since there is no LT factor of $\mathcal{D}_i = 1$. In Fig. 5.5(a) a sub-graph of the FG in Fig. 5.4 is shown. Starting from the left, the first $\mathcal{D}_i = 2$ LT factor is $f_0(t_0, y_0, y_1)$. Assuming input symbol y_1 is randomly chosen, $f_2(t_2, y_0, y_1, y_2)$ has both of its edges removed to become $f_2(t_0, t_2, y_2)$, and $f_1(t_1, y_1, y_4)$ has its edge switched

to become $f_1(t_0, t_1, y_0, y_4)$. Since $f_3(t_3, y_0, y_4)$ is not connected to the chosen input symbol y_1, its structure is not altered even though it shares an input symbol with $f_0(t_0, y_0, y_1)$. The result is illustrated in Fig. 5.5(b).

In this example, the TEP algorithm is successful in producing a $\mathcal{D}_i = 1$ LT factor, i.e., $f_2(t_0, t_2, y_2)$. In fact, the TEP algorithm will only be successful if, and only if, a $\mathcal{D}_i = 3$ LT factor shares both the input symbols of the chosen $\mathcal{D}_i = 2$ LT factor beforehand. However, it is shown in [19] that the probability of producing a $\mathcal{D}_i = 1$ LT factor increases exponentially after each iteration of the TEP algorithm.

5.5.2 Inactivation Decoding

The following algorithm, termed Inactivation Decoding (ID), was developed in order to combine the decoding success rate of GE with the low complexity of BP [20] and is analysed for BECs in [8]. In this work we adapt the algorithm to work in conjunction with FGs.

At the start of the algorithm, all input symbols are considered *active* symbols, where only active symbols are included when calculating the degree $\mathcal{D}_i$ of an LT factor. The algorithm starts by searching for an LT factor of $\mathcal{D}_i = 1$. If found, the column in the LT generation matrix of the output symbol associated with the $\mathcal{D}_i = 1$ LT factor is selected. This column is then XOR'ed with all other columns that have a 1 in the same row.

This is effectively performing GE on the LT generation matrix for the selected output symbol, thus eliminating the active input symbol from all other LT factors. This input symbol of the $\mathcal{D}_i = 1$ LT factor is now considered *recovered* and is accordingly no longer included when calculating $\mathcal{D}_i$; it is thus effectively excluded from the rest of the algorithm.

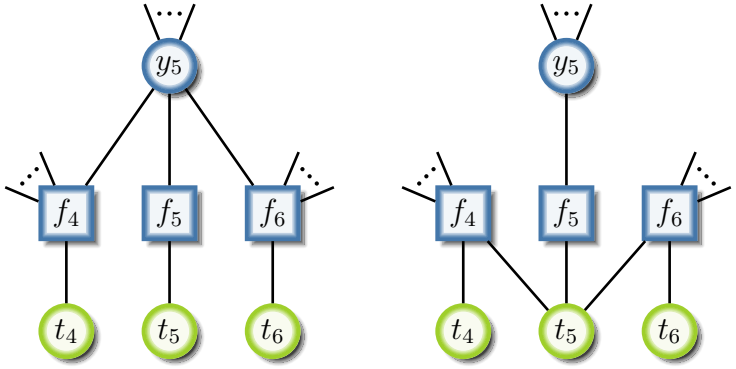
This process continues until no $\mathcal{D}_i = 1$ LT factor exists. When this is the case, the algorithm *inactivates* an input symbol that has not yet been recovered. By doing so the inactivated symbol will be ignored when choosing a factor to eliminate, i.e., all factors connected to the inactivated symbol have their input degree reduced by one.

After a symbol has been inactivated, the algorithm once again searches for a LT factor of $\mathcal{D}_i = 1$. If none is found, another input symbol is inactivated. This continues until all input symbols are either recovered or inactivated. This

implies the necessity for at least as many LT factors and, by extension, output symbols as there are input symbols.

Applying these operations on an FG is somewhat similar to the TEP algorithm. For each input symbol recovered, all edges between it and any LT-factor other than the $\mathcal{D}_i = 1$ LT factor are removed. Subsequently, an edge is added between all output symbols connected to the $\mathcal{D}_i = 1$ LT factor and those LT factors that had the recovered input symbol removed. Once the ID algorithm terminates, BP is applied to the final FG.

An example is depicted in Fig. 5.6, where input symbol y_5 is to be recovered via $f_5(t_5, y_5)$. ID replaces the edge between y_5 and $f_4(t_4, y_2, y_5)$ with the edge between t_5 and $f_4(t_4, t_5, y_2)$. This is also done for the edge between y_5 and $f_6(t_6, y_4, y_5, y_6)$, that is replaced with the edge between t_5 and $f_6(t_5, t_6, y_4, y_6)$.



(a) Before ID (b) After ID

Figure 5.6: A sub-graph of the Raptor code given in Fig. 5.4. Fig. 5.6(a) shows the sub-graph before 1 iteration of ID, with y_5 as the symbol to be recovered. Fig. 5.6(b) shows the sub-graph after ID. This is possible because factor $f_5(t_5, y_5)$ defines the XOR relationship $t_5 \oplus y_5 = 0$, therefore replacing y_5 with t_5 in factors $f_4(y_2, y_5)$ and $f_6(y_4, y_5, y_6)$ retains the validity of their distributions.

In many ways this algorithm is very similar to TEP. In fact, the instance depicted in 5.5 can be viewed as a special case of recovery of y_1, where y_0 is inactivated. However, ID manipulates the graph structure to a greater extent and does not wait for BP to fail before initiating. Therefore both these algorithms end up with vastly different FGs.

5.6 The cluster graph approach

Given a Raptor code with a precode parity-check matrix $\mathbf{H}$ and LT-code generation matrix $\mathbf{G}$ as defined in Section 5.4, it is possible to generate numerous valid CG permutations. It is desirable to have the least possible number of edges in order to minimise the computational complexity of the graph. Unfortunately, finding the optimal solution remains a Non-deterministic Polynomial-time hard (NP-hard) problem [39, p. 418].

A sensible approach for the Raptor code is to generate the CG of the precode before including the clusters from the LT code section. This negates the necessity of reproducing this section of the Raptor code as each output symbol is received. Such a precode CG for the given matrix $\mathbf{H}$ is shown in Fig. 5.7, where the PD of the precode with input symbols $\mathbf{y} = \begin{bmatrix} y_0 & y_1 & \dots & y_6 \end{bmatrix}$ is given as

$$\begin{aligned} P(\mathbf{y}) =& P(y_0)P(y_1)P(y_2)P(y_3)\times \\ & P(y_4|y_0,y_1,y_2)P(y_5|y_1,y_2,y_3)P(y_6|y_0,y_2,y_3). \end{aligned} \tag{5.6.1}$$

To illustrate the equivalent CG, we must express the PD in to form of

$$P(\mathbf{y}) = \frac{\prod_w \phi_w(\mathbf{y}_w)}{\prod_s \psi_s(\mathbf{y}_s)},$$

as given in equation (3.6.2). The product rule is such that

$$P(\mathbf{y}_d|\mathbf{y}_c) = \frac{P(\mathbf{y}_d,\mathbf{y}_c)}{P(\mathbf{y}_c)}$$

and knowing that y_0, y_1, y_2, and y_3 are all independent we have

$$\begin{aligned} P(y_4|y_0,y_1,y_2) &= \frac{P(y_4,y_0,y_1,y_2)}{P(y_0)P(y_1)P(y_2)} \\ P(y_5|y_1,y_2,y_3) &= \frac{P(y_5,y_1,y_2,y_3)}{P(y_1)P(y_2)P(y_3)} \\ P(y_6|y_0,y_2,y_3) &= \frac{P(y_6,y_0,y_1,y_3)}{P(y_0)P(y_1)P(y_3)} \end{aligned}$$

Substituting these equations into equation (5.6.1) we obtain

$$\begin{aligned} P(\mathbf{y}) =& P(y_0)P(y_1)P(y_2)P(y_3)\frac{P(y_4,y_0,y_1,y_2)}{P(y_0)P(y_1)P(y_2)}\times \\ & \frac{P(y_5,y_1,y_2,y_3)}{P(y_1)P(y_2)P(y_3)}\frac{P(y_6,y_0,y_1,y_3)}{P(y_0)P(y_1)P(y_3)} \\ =& \frac{P(y_4,y_0,y_1,y_2)P(y_5,y_1,y_2,y_3)P(y_6,y_0,y_2,y_3)}{P(y_0)P(y_1)P(y_2)^2P(y_3)} \end{aligned}$$

We can now identify the clusters of the graph from the three PDs in the numerator as $\phi_A(y_4,y_0,y_1,y_2)$, $\phi_B(y_5,y_1,y_2,y_3)$, and $\phi_C(y_6,y_0,y_2,y_3)$. Finally, the PDs in the denominator may be grouped into sepsets so that the RIP is maintained, resulting in the following expression:

$$P(\mathbf{y}) = \frac{\phi_A(y_4,y_0,y_1,y_2)\phi_B(y_5,y_1,y_2,y_3)\phi_C(y_6,y_0,y_2,y_3)}{\psi_{AC}(y_0,y_2)\psi_{AB}(y_1,y_2)\psi_{BC}(y_3)}.$$

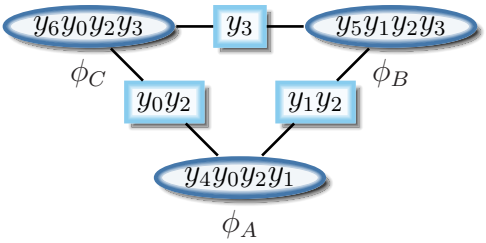


Figure 5.7: An example of a small-scale Raptor precode CG. This precode resembles a (7,4) Hamming code and is to remain constant throughout the decoding process of the Raptor code. Clusters $\phi_B(y_5,y_1,y_2,y_3)$ and $\phi_C(y_6,y_0,y_2,y_3)$ only have the sepset $\psi(y_3)$ in order to satisfy the RIP.

Thereafter, each LT cluster may be added to the precode CG independently as its corresponding output symbol is received. Moreover, each LT cluster is added in such a way that each of its input symbols is included in one, and only one, sepset that is connected to the LT cluster itself. Assuming that the RIP of the precode CG is satisfied, adding the LT clusters in such a manner is guaranteed to preserve the RIP, since each LT cluster will form an end point of the RIP for each symbol in its scope. In essence this approach produces a somewhat starlike topographical graph.

The complete CG, with a precode parity-check matrix $\mathbf{H}$ and LT code-generation matrix $\mathbf{G}$, as generated using the above approach, is given in

Fig. 5.8. Given that $\mathbf{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_7 \end{bmatrix}$, the PD of the LT cluster section can be expressed as

$$\begin{aligned} P(\mathbf{t}) =& P(t_0|y_0, y_1)P(t_1|y_1, y_4)P(t_2|y_0, y_1, y_2)P(t_3|y_0, y_4)\times \\ & P(t_4|y_2, y_5)P(t_5|y_5)P(t_6|y_5, y_6)P(t_7|y_4y_6). \end{aligned}$$

Again, using the product rule we have

$$\begin{aligned} P(\mathbf{t}) =& \frac{P(t_0, y_0, y_1)}{P(y_0)P(y_1)} \times \frac{P(t_1, y_1, y_4)}{P(y_1)P(y_4)} \times \frac{P(t_2, y_0, y_1, y_2)}{P(y_0)P(y_1)P(y_2)} \times \frac{P(t_3, y_0, y_4)}{P(y_0)P(y_4)}\times \\ & \frac{P(t_4, y_2, y_5)}{P(y_2)P(y_5)} \times \frac{P(t_5, y_5)}{P(y_5)} \times \frac{P(t_6, y_5, y_6)}{P(y_5)P(y_6)} \times \frac{P(t_7, y_4, y_6)}{P(y_4)P(y_6)} \end{aligned}$$

and identifying the clusters and sepsets the result is as follows:

$$\begin{aligned} P(\mathbf{t}) =& \frac{\phi_0(t_0, y_0, y_1)}{\psi_{A0}(y_0, y_1)} \times \frac{\phi_1(t_1, y_1, y_4)}{\psi_{A1}(y_1, y_4)} \times \frac{\phi_2(t_2, y_0, y_1, y_2)}{\psi_{A2}(y_0, y_1, y_2)} \times \frac{\phi_3(t_3, y_0, y_4)}{\psi_{A3}(y_0, y_4)}\times \\ & \frac{\phi_4(t_4, y_2, y_5)}{\psi_{B4}(y_2, y_5)} \times \frac{\phi_5(t_5, y_5)}{\psi_{B5}(y_5)} \times \frac{\phi_6(t_6, y_5, y_6)}{\psi_{A6}(y_5)\psi_{C6}(y_6)} \times \frac{\phi_7(t_7, y_4, y_6)}{\psi_{A7}(y_4)\psi_{C7}(y_6)}. \end{aligned}$$

Thus the CG in Fig. 5.8 represents the PD $P(\mathbf{t}, \mathbf{y})$. From this it can clearly be seen that each LT cluster can be added to the graph independently whilst maintaining the RIP constraint.

Decoding the CG requires the following algorithm:

1. Populate the PD of all clusters to describe the relationship of the variables such that

$$\phi_m(\mathbf{t}_m, \mathbf{y}_m) = \begin{cases} 1 & \text{for } \left(\bigoplus_{t\in\mathbf{t}_m} \mathbf{t}_m\right) \oplus \left(\bigoplus_{y\in\mathbf{y}_m} \mathbf{y}_m\right) = 0 \\ 0 & \text{otherwise} \end{cases}$$

   for all m. Note that $\mathbf{t}_m$ may be an empty set, i.e., $\mathbf{t}_m = \varnothing$

2. Factor in the output symbol PDs $P(t_n|r_n = \mathcal{V})$ for all n where r_n is the observed symbol corresponding to t_n and $\mathcal{V}$ is the observation made. That is, multiply each output symbol's PDs once, and only once, with any cluster that includes the symbol in its scope.

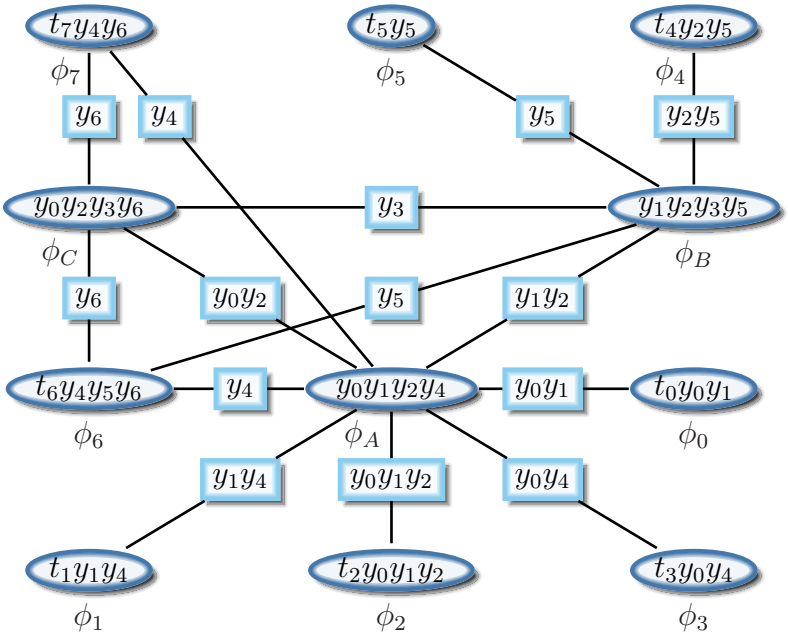


Figure 5.8: An example of a small-scale Raptor code CG. It includes the precode that consists of the clusters $\phi_A(y_4, y_0, y_2, y_1)$, $\phi_B(y_5, y_1, y_2, y_3)$, and $\phi_C(y_6, y_0, y_2, y_3)$, as well as the LT code that makes up the rest of the clusters. This permutation of the CG is due to the initial construction of the precode, followed by adding the LT clusters one at a time.

3. Update the message from cluster $\phi_m(\mathbf{x}_m)$ to cluster $\phi_n(\mathbf{x}_n)$, for all m and n, using (4.7.1) such that:

$$\mu_{\phi_m \to \phi_n}(\mathbf{x}_S) = \sum_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \left(\phi_m(\mathbf{x}_m) \prod_{\substack{\phi_i = \\ \boldsymbol{\phi}_m \setminus \phi_n}} \mu_{\phi_i \to \phi_m}(\mathbf{x}_T) \right),$$

where $\mathbf{x}_S \subseteq (\mathbf{x}_m \cap \mathbf{x}_n)$, $\mathbf{x}_T \subseteq (\mathbf{x}_i \cap \mathbf{x}_m)$, $\mathbf{x_m} \subseteq \begin{bmatrix} \mathbf{t} & \mathbf{y} \end{bmatrix}$, and $\mathbf{x_n} \subseteq \begin{bmatrix} \mathbf{t} & \mathbf{y} \end{bmatrix}$.

4. Calculate the marginal PDs of the input symbols, i.e.,

$$P(y_n) = \sum_{\substack{y_i = \\ \mathbf{y}_m \setminus y_n}} \sum_{\substack{t_i = \\ \mathbf{t}_m}} \phi_m(\mathbf{t}_m, \mathbf{y}_m)$$

for all n and any m where $y_n \in \mathbf{y}_m$.

5. Repeat Item 3 and Item 4 until

$$\frac{1}{\mathcal{N}} \sum_{y_n = \mathbf{y}} \Big(L(y_n)_i - L(y_n)_{i-1} \Big) < \mathcal{Q}.$$

or

$$i \geq \mathcal{I}_{max}.$$

That is, choose some threshold value $\mathcal{Q}$ such that convergence is achieved once the difference in the average of the LLR of all input symbol marginal PDs, from BP pass $i-1$ to BP pass i, do not exceed $\mathcal{Q}$. $\mathcal{I}_{max}$ is the selected maximum number of BP iterations allowed per decoding attempt.

6. If the number of iterations i exceeds the maximum allowed number $\mathcal{I}_{max}$, wait for a new observation to be made and add its corresponding cluster to the CG. Thereafter, restart the algorithm from Item 1. If $i < \mathcal{I}_{max}$, a local maximum has been reached.

5.7 The junction tree algorithm

The final approach to decoding the Raptor code is the JT: a subset of the CG topology. It is constructed using the VE algorithm, which will *always* produce treelike structured graphs, allowing for exact inference.

The order of elimination of the VE algorithm is critical for the resulting computational complexity of the graph. Finding the optimal elimination order is an NP-hard problem and thus in practice it is done using the *greedy search* algorithm, where the decision of which variable to eliminate is done *on-the-fly*. The *min-weight* cost function is used, where the cost of each elimination is based on the cardinality ($|X|$) of the PD that is associated with the resulting cluster due to the elimination.

Since all the output symbols will always only belong to a single cluster, their clusters will be eliminated first in order to maintain the lowest computational complexity possible. Furthermore, the order in which the output clusters are eliminated is irrelevant as they are all structured independently. As a result, the output clusters will all be leaf nodes of the JT.

Where we were able to keep the precode section of the graph constant for the CG, for the JT it is necessary to restructure the entire graph for each new output cluster introduced, since it may influence the cost function.

Fig. 5.9 shows an example of a JT for the instance introduced in Section 5.4 with parity-check matrix $\mathbf{H}$ and LT code-generation matrix $\mathbf{G}$, as compiled by the method given here. Given that $\mathbf{y} = \begin{bmatrix} y_0 & y_1 & \dots & y_6 \end{bmatrix}$ and $\mathbf{t} = \begin{bmatrix} t_0 & t_1 & \dots & t_7 \end{bmatrix}$, the resulting elimination order is

$$\langle t_0, t_1, t_2, t_3, t_4, t_5, t_6, t_7, y_4, y_0, y_1 \rangle.$$

Symbols y_2, y_3, y_5, and y_6 are not included in the VE as they are all contained in the scope of the cluster produced by the final elimination, that of y_1. The resulting clusters ϕ_m, for $m = 0, 1, \ldots, 10$, are numbered in the order they were obtained. That is, cluster ϕ_m was obtained from the m^{th} elimination.

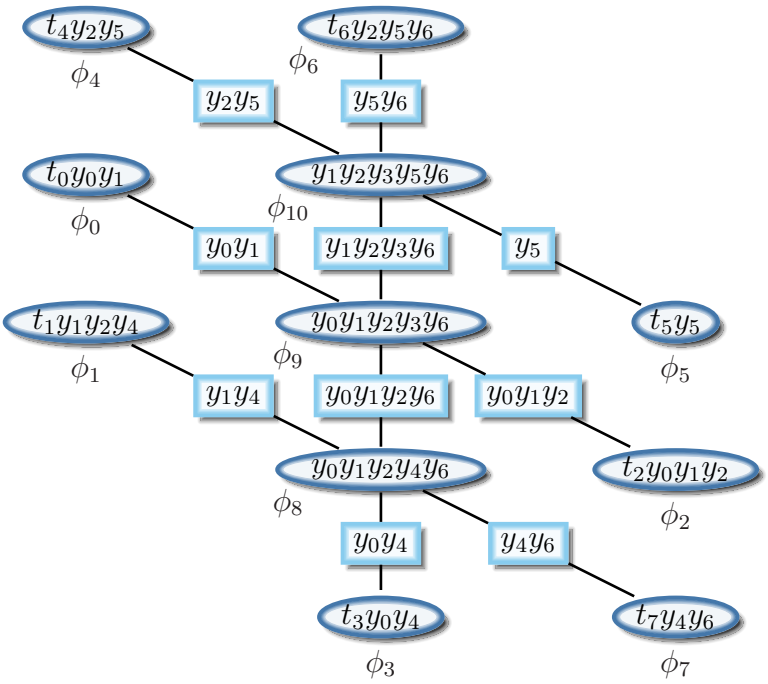


Figure 5.9: An example of a small-scale Raptor code JT. It includes both the precode and LT code clusters. This permutation of the JT is due to the elimination order of $\langle \mathbf{y}, y_4, y_0, y_1 \rangle$. The RIP is satisfied for all symbols.

The process of decoding a JT is very similar to that of a CG, except that exact inference is done and thus no iteration is required. The decoding algorithm follows accordingly:

1. Populate the PD of all clusters to describe the relationship of the variables such that

$$\phi_m(\mathbf{t}_m, \mathbf{y}_m) = \begin{cases} 1 & \text{for } \left(\bigoplus_{t \in \mathbf{t}_m} \mathbf{t}_m\right) \oplus \left(\bigoplus_{y \in \mathbf{y}_m} \mathbf{y}_m\right) = 0 \\ 0 & \text{otherwise} \end{cases}$$

   for all m. Note that $\mathbf{t}_m$ may be an empty set, i.e., $\mathbf{t}_m = \varnothing$

2. Factor in the output symbol PDs $P(t_n | r_n = \mathcal{V})$, for all n where r_n is the observed symbol corresponding to t_n and $\mathcal{V}$ is the observation made.

That is, multiply each ouput symbol's PDs once, and only once, with any cluster that includes the symbol in its scope.

3. Update the message from cluster $\phi_m(\mathbf{x}_m)$ to cluster $\phi_n(\mathbf{x}_n)$, for all m and n, using (4.7.1) such that:

$$\mu_{\phi_m \to \phi_n}(\mathbf{x}_S) = \sum_{\substack{x_i = \\ \mathbf{x}_m \setminus x_n}} \left(\phi_m(\mathbf{x}_m) \prod_{\substack{\phi_i = \\ \boldsymbol{\phi}_m \setminus \phi_n}} \mu_{\phi_i \to \phi_m}(\mathbf{x}_T) \right),$$

where $\mathbf{x}_S \subseteq (\mathbf{x}_m \cap \mathbf{x}_n)$, $\mathbf{x}_T \subseteq (\mathbf{x}_i \cap \mathbf{x}_m)$, $\mathbf{x_m} \subseteq \begin{bmatrix} \mathbf{t} & \mathbf{y} \end{bmatrix}$, and $\mathbf{x_n} \subseteq \begin{bmatrix} \mathbf{t} & \mathbf{y} \end{bmatrix}$.

4. Calculate the marginal PDs of the input symbols, i.e.,

$$P(y_n) = \sum_{\substack{y_i = \\ \mathbf{y}_m \setminus y_n}} \sum_{\substack{t_i = \\ \mathbf{t}_m}} \phi_m(\mathbf{t}_m, \mathbf{y}_m)$$

for all n and any m where $y_n \in \mathbf{y}_m$.

5.8 Summary

This chapter focused on how the theory covered in this work is applied. More specifically, we saw how PGMs and BP are applied to the Raptor code. We also considered the implementation of the FG, the CG, and the JT.

Initially, the general application of PGMs on FEC codes is explained in Section 5.2. This mainly concerns the observed values, which do not pertain to any of the variables included in the original graph and must therefore be added as additional nodes. By applying absolute inference to these new nodes, the complexity of the graph can be reduced before running the iterative process of loopy BP.

The standardised R10 code was covered next in Section 5.3, and is chosen for its lower complexity as compared to the RQ code. The R10 is a systematic code designed to encode up to 2^{13} source symbols and 2^{16} output symbols and is designed specifically for the application of the ID. The precode's layout is given in (5.3.2) and is designed to maximise the likelihood that sufficient rank is generated in the code for a given number of output symbols so that the decoding will be successful. The LT section of the R10 code is generated using the degree distribution in Table 5.1. Throughout the chapter the same example was used to illustrate the decoding process.

Figure 5.4 depicted the FG of an R10 code as 3 top-to-bottom stages in Section 5.4. Decoding any practical Raptor code FG is done using approximate BP, where messages are iterated across the graph until convergence is achieved or a maximum number of iterations is reached. If the maximum number of iterations is reached, the decoding has failed.

One fundamental problem with fountain codes is that enough input degree 1 ($\mathcal{D}_i = 1$) LT factors are required in order to initialise the BP algorithm and successfully decode the code. As was explained in Section 5.5, with very little to no increase in complexity we can use the relationship of the input and output symbols to change the structure of the FG in order to obtain the necessary $\mathcal{D}_i = 1$ LT factors. Thus if BP fails, one of two algorithms may be used in an attempt to improve the recovery rate, i.e., TEP or ID.

TEP was covered in Section 5.5.1, which was proposed and analysed in [19]. The algorithm is based on the observation that the relationships of the symbols, as defined by a given LT factor, are simultaneous XOR equations. By rearranging these equations, the algorithm changes the graph structure of an FG. This is useful when decoding graphs with an LT section that has an insufficient degree distribution. Every time a BP decoding attempt fails, the graph alteration process will be repeated until either BP is successful, or no more $\mathcal{D}_i = 2$ LT factors remain.

In Section 5.5.2 we covered that ID was developed in order to combine the decoding success rate of GE with the low complexity of BP [20]. The algorithm starts by searching for an LT factor of $\mathcal{D}_i = 1$. If found, the column in the LT generation matrix of the output symbol associated with the $\mathcal{D}_i = 1$ LT factor is selected. This column is then XOR'ed with all other columns that have a 1 in the same row.

The corresponding input symbol of the $\mathcal{D}_i = 1$ LT factor is now considered *recovered*. If no $\mathcal{D}_i = 1$ LT factor exists, the algorithm *inactivates* an input symbol that has not yet been recovered and once again searches for a LT factor of $\mathcal{D}_i = 1$. This continues until all input variables are either recovered or inactivated. Finally, BP is applied to the newly formed FG.

Section 5.6 covered the application of the CG to the Raptor code. The CG of a Raptor code is initially generated using only the precode, negating the necessity to reproduce this section of the Raptor code as each output symbol is received. Each LT cluster may then be added on-the-fly without altering the existing graph. Similar to the FG, a Raptor code CG is decoded using

approximate BP.

Finally, Section 5.7 saw the JT constructed using the VE algorithm, which will *always* produce treelike structured graphs. The order of elimination is done using the *greedy search* algorithm along with the *min-weight* cost function. The process of decoding a JT is very similar to that of a CG, except that exact inference is done and thus no iteration is required.

Chapter 6

Simulations and Results

6.1 Introduction

This chapter covers the results of the simulations done to test and confirm the topics and theories discussed within this thesis.

First we determine whether the Raptor code is a viable code to use for all channel models covered in this work, namely the Binary Erasure Channel (BEC), the Binary Symmetric Channel (BSC), and the Binary Additive White Gaussian Noise Channel (BAWGNC). In doing so, a comparison is made of the Bit Error Rates (BERs) of the Raptor code over the channel models in Section 6.2.

It is shown in Section 6.3 that the transmission rate of the Raptor code approaches a point close to channel capacity as the number of input symbols $\mathcal{K}$ goes to infinity. That is, the transmission rate of the Raptor code is close to the channel capacity for large values of $\mathcal{K}$.

In Section 6.4 we show that the BER of the Raptor code is not limited by an error floor, whereas the Luby Transform (LT) code is. This illustrates that the Raptor code is an improvement on the LT code.

Thereafter we compare the graph altering algorithms, Tree-structure Expectation Propagation (TEP) and Inactivation Decoding (ID), against the tanh-rule loopy Belief Propagation (BP) algorithm in Section 6.5. This is done by comparing the BERs of the three algorithms.

The BERs of the Factor Graph (FG), Cluster graph (CG), and Junction Tree (JT) when applied to a Raptor code are compared in Section 6.6. Furthermore, we consider the difference in computational complexities of these

Probabilistic Graphical Models (PGMs) in this application. This is done by comparing the number of messages passed before successful decoding is achieved.

In all applicable cases, the *overhead* is calculated as a fraction of the total symbols above the required number to communicate at channel capacity $\text{Cap}(\mathcal{C})$. That is, should $\text{Cap}(\mathcal{C}) = 0.5$ with $\mathcal{K}$ source symbols, it would require $\mathcal{M} = 2\mathcal{K}$ output symbols to communicate reliably. An overhead of 0.3 would therefore imply that a total of $\mathcal{M} = 2\mathcal{K}(1 + 0.3) = 2.4\mathcal{K}$ output symbols were transmitted.

Furthermore, for all simulations it was required that at least a 100-bit error must be detected for each data point. Thus, a minimum of $N = 100/(\text{BER} \times \mathcal{K})$ total decoding instances were simulated for each overhead step where the BER $\geq 1 \times 10^{-3}$. For BER $< 1 \times 10^{-3}$, a minimum of 100 decoding instances were simulated. For example, if the BER is 3.20×10^{-4} for $\mathcal{K} = 1000$, a minimum of

$$\frac{100}{\text{BER} \times \mathcal{N}} = \frac{100}{(3.20 \times 10^{-4})(1072)} = 2986$$

decoding instances were simulated.

6.2 BER analysis of BP on FGs over BEC, BSC, and BAWGNC

In this section, we focus on analysing whether the Raptor code performance against the channel capacity is similar for the three channel models covered in this thesis. This is done in terms of BER against the overhead.

The standardised Raptor 10 (R10) Raptor code was used for these simulations and the encoder and the decoding algorithms were implemented using Python. Each simulation was executed with a source symbol set size of $\mathcal{K} = 1000$, where each symbol is only 1 bit long. This implies that the number of input symbols was $\mathcal{N} = 1072$.

For all three the channel models, the channel capacity was set to $\text{Cap}(\mathcal{C}) \approx 0.5$. That is, the BEC has an erasure probability of $\epsilon = 0.5$, the BSC has a bit-flip probability $\rho = 0.11$, and the BAWGNC has a Signal-to-Noise Ratio (SNR) of $0.374dB$.

The decoding was done using the tanh-rule loopy BP over FGs with a message-flooding schedule. The overhead is incremented from -0.2 to 0.6 in

steps of 0.05. $\mathcal{I}_{max} = 500$ is the maximum number of iterations allowed for each decoding simulation. If this threshold was reached, the decoding was terminated and the last iteration's marginal Probability Distributions (PDs) was used to check the correctness of the answer.

A threshold value $\mathcal{Q} = 0.0002$ was used to determine if convergence was achieved. This threshold is compared to the average difference in the Log-Likelihood Ratios (LLRs) of the marginal PD of the input symbols $\mathbf{y}$ for the most recent and the previous iterations. That is

$$\frac{1}{\mathcal{N}} \sum_{y_n=\mathbf{y}} \left(L(y_n)_i - L(y_n)_{i-1} \right) < \mathcal{Q}$$

where $L(y_n)_i$ is the LLR of input symbol y_n of iteration number i.

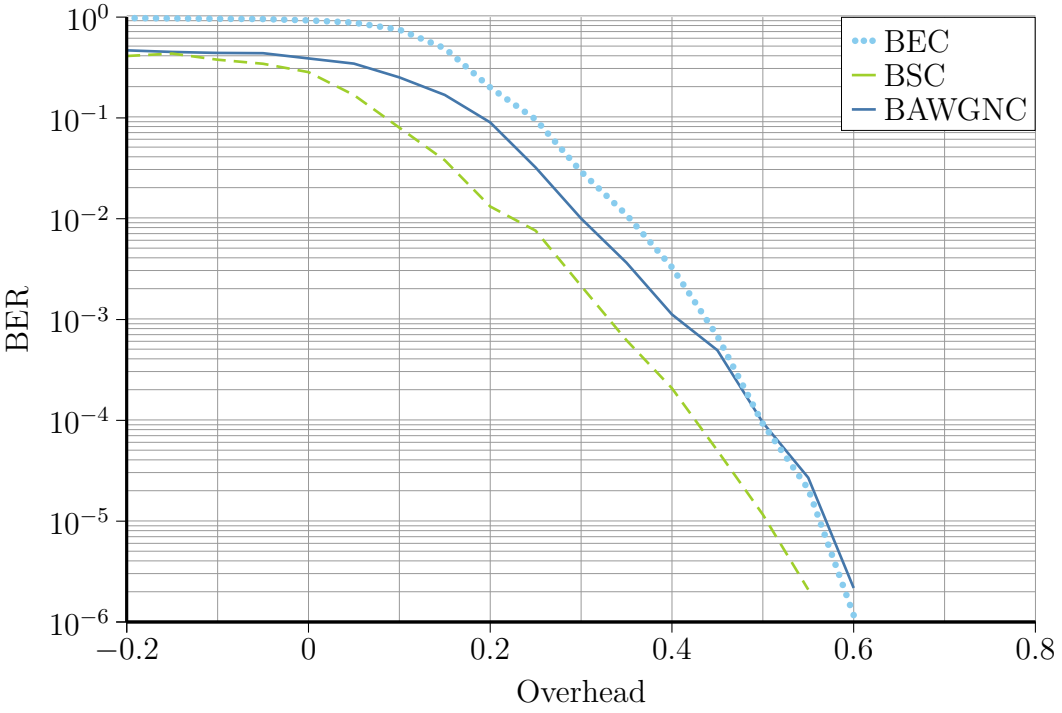


Figure 6.1: The standardised R10 Raptor code was simulated with a source symbol set size of $\mathcal{K} = 1000$. Here we see its performance against the channel capacity of the BEC, BSC, and BAWGNC models. This is done in terms of BER against the overhead. From this figure it can be seen that, for all three channel models, the R10 code has a BER drop-off approximately at the same overhead.

To analyse the decoding success rates of the channels, we look at their BER drop-offs. A BER drop-off is the point at which the BER curve, as compared to the overhead, starts to decline rapidly for each unit the overhead

increases. From Fig. 6.1 it can be seen that, for all three channel models, the R10 code has a BER drop-off approximately at the same overhead, that is, at approximately 0.1 overhead. This implies that the Raptor code is a viable code to use for all three models.

Indeed, it is notable from these results that for an overhead less that 0.5, both the BSC and BAWGNC outperform the BEC. This is unexpected since the R10 code was designed specifically for the BEC. This difference may be attributed to two potential factors:

1. The decoding of information passed over a BEC applies deterministic inference, whereas information passed over a BSC or a BAWGNC applies probabilistic inference. The difference in the ability of these two types of inference to propagate information across a graph with insufficient information may lead to more mistakes for the deterministic case.

2. Although unlikely, the decoding approach chosen may favour the behaviour of one channel model over another. For example, the message-passing schedule may influence the BER of the BEC more than the other channel models.

The superior BER curve of the BSC as compared to the BAWGNC is a somewhat surprising result. However, this may largely be attributed to the approximation of the channel capacities of both the BSC and the BAWGNC. That is, for the simulations done in this section, $\text{Cap}(\mathcal{C}_{BSC}) > \text{Cap}(\mathcal{C}_{BAWGNC})$ due to approximation errors.

6.3 BER analysis of the Raptor code and the input symbol set size

It was stated in Section 2.6.2 that a postulation was made by Shokrollahi *et al* in [20] for the random linear fountain code. This stated that, for a small overhead o such that $\mathcal{M} = \mathcal{K} + o$, the failure probability δ has the upper bound of 2^{-o}. where $\mathcal{K}$ is the number of input symbols and $\mathcal{M}$ is the number of output symbols. Furthermore, this upper bound is found to be independent of the size of $\mathcal{K}$. Thus, for large values of $\mathcal{K}$, an overhead of relatively trivial size is required for a high probability of success.

In this section, we show that the standardised R10 Raptor code exhibits similar behaviour. This is done by analysing BER of the code against the overhead for input-symbol set sizes of $\mathcal{K} = 100, 1000, 8192$.

For these simulations, the encoder and the decoding algorithms were implemented using Python. Each simulation was executed where each symbol was only 1 bit long. All three simulations where executed over the BSC with a channel capacity of $\text{Cap}(\mathcal{C}_{BSC}) \approx 0.5$.

The decoding was done using the tanh-rule loopy BP over FGs with a message-flooding schedule. The overhead is incremented from -0.1 to 0.9 in steps of 0.05. $\mathcal{I}_{max} = 500$ is the maximum number of iterations allowed for each decoding simulation. If this threshold was reached, the decoding was terminated and the last iteration's marginal PDs used to check the correctness of the answer.

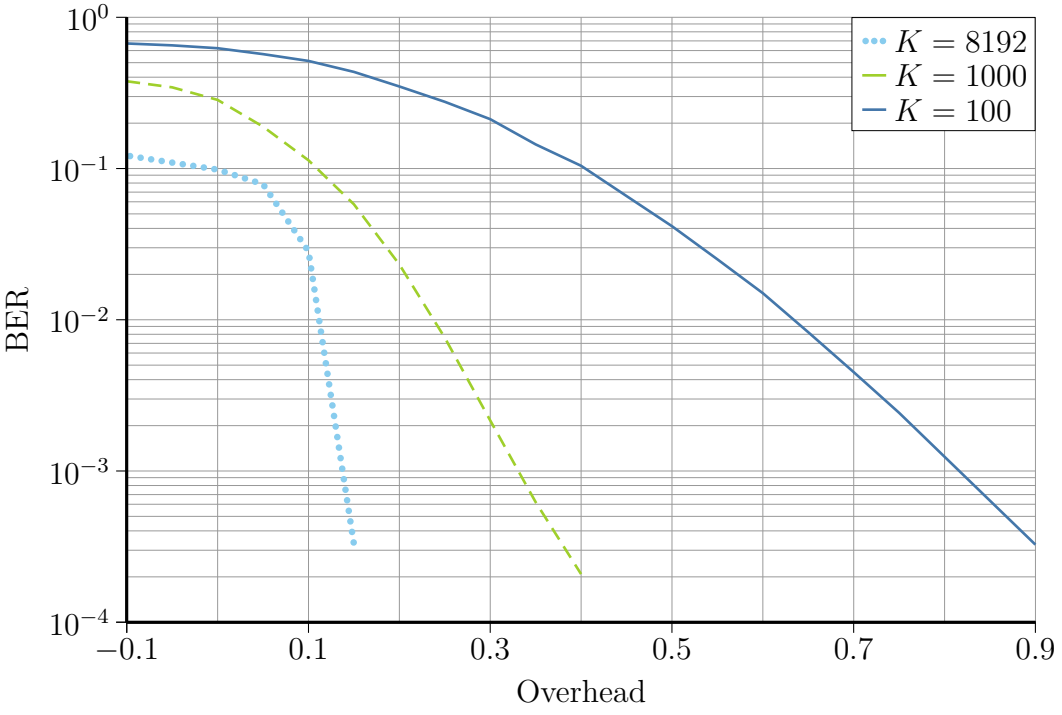


Figure 6.2: The standardised R10 Raptor code was simulated over the BSC, using FGs and with source-symbol set sizes of $\mathcal{K} = 100, 1000, 8192$. It can be observed that, as $\mathcal{K} \to \infty$, the BER drop-off approaches the channel capacity.

A threshold value $\mathcal{Q} = 0.0002$ was used to determine if convergence was achieved. This threshold is compared to the average difference in the LLRs of the marginal PD of the input symbols $\mathbf{y}$ for the most recent and the previous

iterations. That is,

$$\frac{1}{\mathcal{N}} \sum_{y_n=\mathbf{y}} \left(L(y_n)_i - L(y_n)_{i-1} \right) < \mathcal{Q}$$

where $L(y_n)_i$ is the LLR of input symbol y_n of iteration number i.

Figure 6.2 shows the results of these simulations. From this it can be observed that as the number of input symbols $\mathcal{K}$ goes to infinity, the BER drop-off approaches a point close to the channel capacity.

Even though the R10 code is only designed for input-symbol set sizes of up to 2^{13}, it is safe to assume that other Raptor codes (, e.g., the RaptorQ (RQ) code) would do so as well, since both the random linear fountain code and the R10 code show similar behaviour. Accordingly, we may conclude that the transmission rate of the Raptor code is close to the channel capacity for large values of $\mathcal{K}$.

6.4 A comparison between the BERs of the Raptor code and the LT code

The improvements of the Raptor code with respect to the LT code have been shown in [17, 46]. In this section we confirm that this is the case for the BSC by comparing the BER of the Raptor code and LT code against the overhead.

The standardised R10 code was used for these simulations, and the encoder and decoding algorithms were implemented using Python. The LT code used is equivalent to the R10 code without a precode. That is, the LT code used has a degree distribution as defined in Table 5.1. Each simulation was executed with a source symbol set size of $\mathcal{K} = 1000$, where each symbol is only 1 bit long. This implies that the number of input symbols was $\mathcal{N} = 1072$.

The BSC had a bit-flip probability $\rho = 0.11$. This implies that the channel capacity was set to $\text{Cap}(\mathcal{C}_{BSC}) \approx 0.5$.

The decoding was done using the tanh-rule loopy BP with a message-flooding schedule. The overhead is incremented from 0 to 1 in steps of 0.05. $\mathcal{I}_{max} = 500$ is the maximum number of iterations allowed for each decoding simulation. If this threshold was reached, the decoding was terminated and the last iteration's marginal PDs used to check the correctness of the answer.

A threshold value $\mathcal{Q} = 0.0002$ was used to determine if convergence was achieved. This threshold is compared to the average difference in the LLRs of

the marginal PD of the input symbols $\mathbf{y}$ for the most recent and the previous iterations. That is

$$\frac{1}{\mathcal{N}} \sum_{y_n=\mathbf{y}} \left(L(y_n)_i - L(y_n)_{i-1} \right) < \mathcal{Q}$$

where $L(y_n)_i$ is the LLR of input symbol y_n of iteration number i.

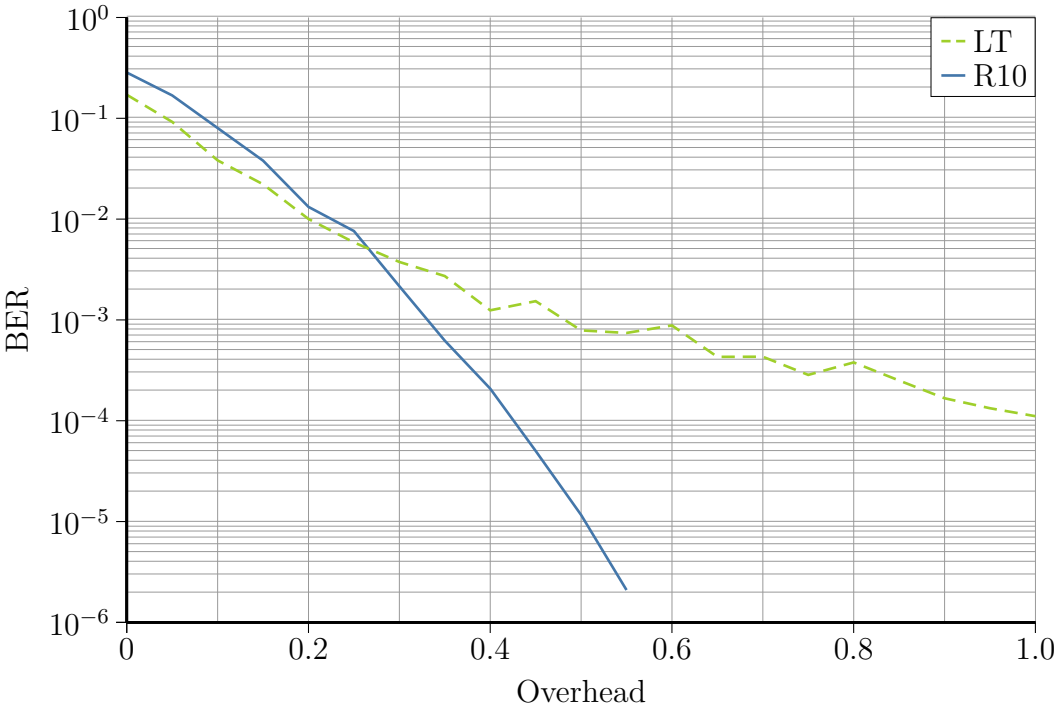


Figure 6.3: This figure depicts the BERs of both the R10 and the LT code against the overhead. It shows that the R10 is not limited by an error floor, whereas the LT code is. This result illustrates that the Raptor code successfully enables the decoding of the fraction of source symbols that the LT code does not encode.

In Fig. 6.3 we show that the BER of the R10 is not limited by an error floor, whereas the LT code is. The error floor of the LT code is a side effect of its degree distribution, which tends to leave a small fraction of the source data unencoded. The receiver will thus never be able to decode these fractions of the source data. This result thus illustrates that the Raptor code successfully enables the decoding of those fractions.

6.5 BER analysis of BP, TEP and ID decoding on FGs

The performance of the graph-altering algorithms, TEP and ID, is analysed in this section by comparing their BERs against the overhead.

These algorithms were tested on the R10 and compared against the tanh-rule loopy BP algorithm. These test simulations were run over the BSC, using FGs to decode the R10 code with a source data set size of $\mathcal{K} = 100$, where each symbol is only 1 bit long. The decoding algorithms were implemented using Python. The BSC had a bit-flip probability $\rho = 0.11$. This implies that the channel capacity was set to $\text{Cap}(\mathcal{C}_{BSC}) \approx 0.5$.

In all three decoding algorithms, the message-flooding schedule was applied. The overhead is incremented from 0 to 1.2 in steps of 0.05. $\mathcal{I}_{max} = 500$ is the maximum number of iterations allowed for each decoding simulation. If this threshold was reached, the decoding was terminated and the last iteration's marginal PDs used to check the correctness of the answer.

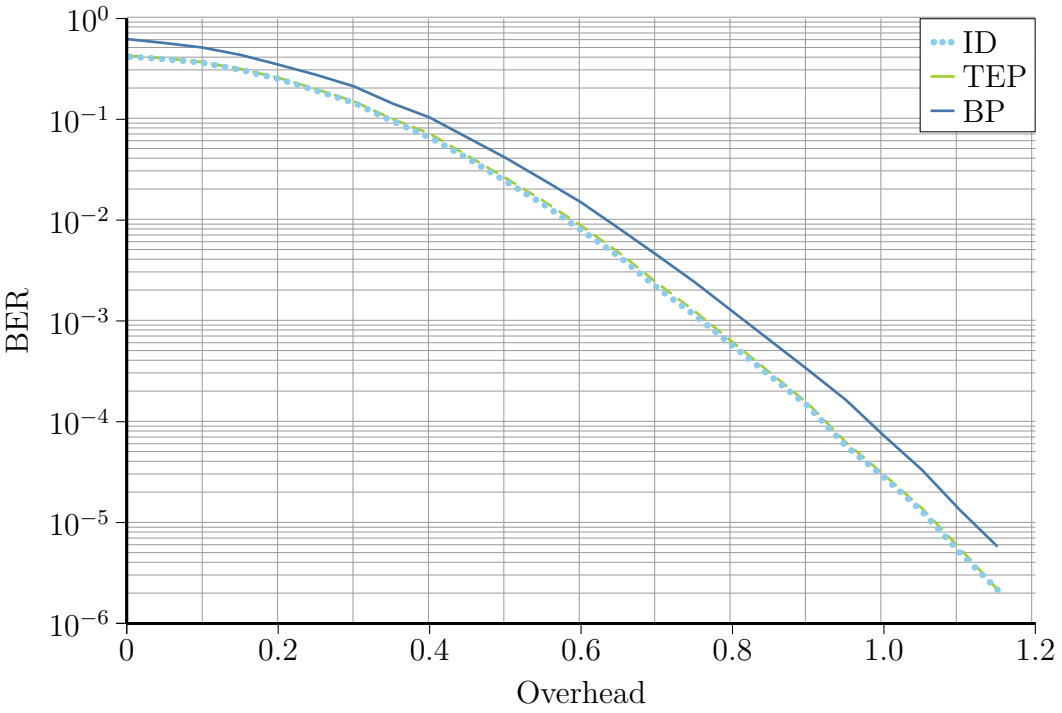


Figure 6.4: The standardised R10 Raptor code was simulated with a source symbol set size of $\mathcal{K} = 100$. Here we see that both the ID and TEP algorithms showed improvements in their BER with respect to the tanh-rule loopy BP algorithm. However, the performance increase of ID as compared to the TEP is neglectably small.

A threshold value $\mathcal{Q} = 0.0002$ was used to determine if convergence was achieved. This threshold is compared to the average difference in the LLRs of the marginal PD of the input symbols $\mathbf{y}$ for the most recent and the previous iterations. That is

$$\frac{1}{\mathcal{N}} \sum_{y_n=\mathbf{y}} \Big(L(y_n)_i - L(y_n)_{i-1} \Big) < \mathcal{Q}$$

where $L(y_n)_i$ is the LLR of input symbol y_n of iteration number i.

As depicted in Fig. 6.4, it was found that both the ID and TEP algorithms showed marginal improvements in their BER with respect to the tanh-rule loopy BP algorithm. However, these improvements are only significant for small code sizes. For $\mathcal{K} \geq 300$, almost no improvement is observed.

This does not match the results obtained by M. A. Guede in [24] that show improvements of $\mathcal{K} = 100$ to $\mathcal{K} = 1000$ for the TEP. However, those simulations were executed using an LT code with a robust soliton degree distribution over the BEC.

The TEP is designed to improve the decoding success rate of an LT code by repairing any possible degree 1 output symbol deficiency. Thus, we may conclude that the LT section of the R10 code does not suffer from a deficiency in degree 1 output symbols for large enough code sizes. This may be contributed to the fact that the degree distribution of the R10 code is weakened, i.e., it has a lower average output symbol degree with respect to the robust Soliton distribution.

The performance increase of ID as compared to the TEP is neglectably small. Thus the TEP is considered preferable due to its lower computational complexity compared to that of ID.

6.6 BER analysis of PGMs with BP decoding

In this section we compare the BERs of the PGMs we focus on in this thesis, namely the FG, CG, and JT.

Since we cannot make use of the tanh-rule for the CG and JT, it was decided to use a different Raptor code than the R10 in order to reduce the computational complexity of the graphs. For this comparison a left-regular Low-Density Parity-Check (LDPC) code was used for the precode, and the maximum output symbol degree was limited to 24, such that the resulting output degree distribution is as given in Table 6.1.

Table 6.1: The degree distribution of the R10 code, with an average degree of 4.63. This is designed to work well with ID.

$\Omega(\mathcal{D})$	0.0098	0.4590	0.2110	0.1134	0.1113	0.0799	0.0156
$\mathcal{D}$	1	2	3	4	10	11	24

The decoding of the graphs was done using the loopy BP algorithm. These test simulations were run over the BSC, with a source data set size of $\mathcal{K} = 100$, where each symbol is only 1 bit long. The encoding and decoding algorithms were implemented using c++. The BSC had a bit-flip probability $\rho = 0.11$. This implies that the channel capacity was set to $\text{Cap}(\mathcal{C}_{BSC}) \approx 0.5$.

In all three simulations the sequential message-passing schedule was applied. The overhead is incremented in steps of 0.2. The maximum number of message allowed to be passed for each decoding simulation is $3000\mathcal{W}$, where $\mathcal{W}$ is the number of clusters in the graph. If this threshold was reached, the decoding was terminated and the last iteration's marginal PDs used to check the correctness of the answer.

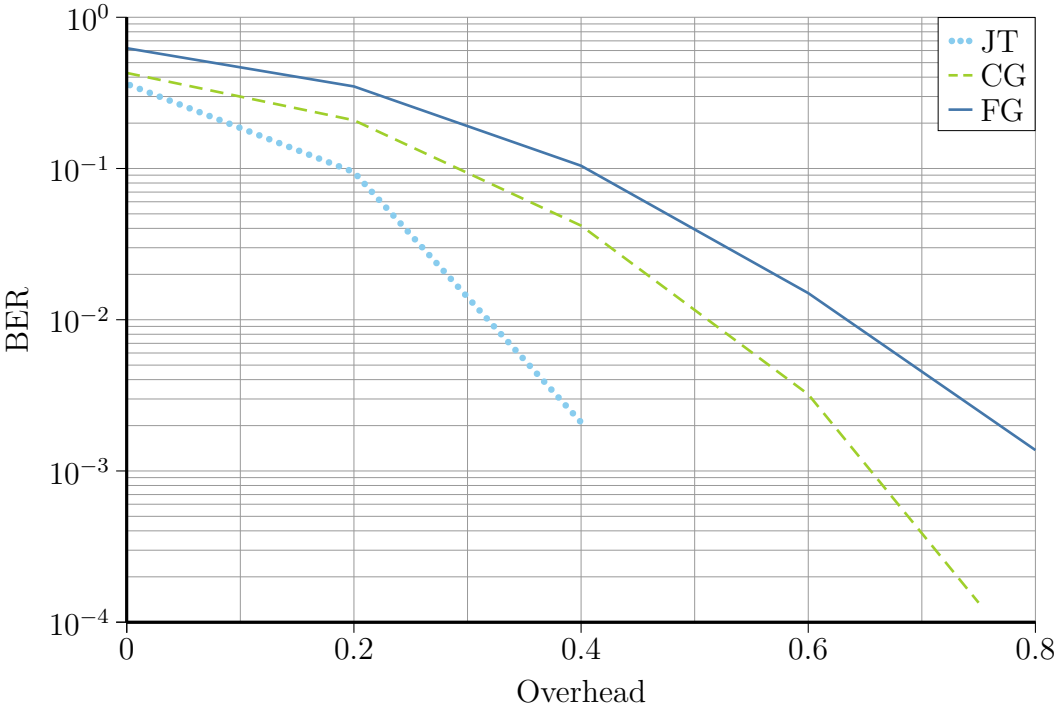


Figure 6.5: The Raptor code was simulated with a source symbol set size of $\mathcal{K} = 100$ and decoded using the FG, the CG, and the JT. Here we see that the BER of the CG is better than that of the FG. The BER of the JT outperforms the BER of both the FG and the CG.

A threshold value $\mathcal{Q} = 0.001$ was used to determine if convergence was achieved. This threshold is compared to the average difference in the messages passed for the most recent and the previous iterations. That is

$$\frac{1}{N}\sum_{\phi_n=\boldsymbol{\phi}_n}\sum_{\phi_m=\boldsymbol{\phi}_m}\left(\mu_{\phi_m\to\phi_n}(\mathbf{x}_m\cap\mathbf{x}_n)_i-\mu_{\phi_m\to\phi_n}(\mathbf{x}_m\cap\mathbf{x}_n)_{i-1}\right)<\mathcal{Q}$$

where $\mu_{\phi_m\to\phi_n}(\mathbf{x}_m\cap\mathbf{x}_n)_i$ is a message passed in iteration number i.

In Fig. 6.5 it can be seen that the BER of the CG is better than that of the FG. The BER of the JT outperforms the of both the FG and the CG.

The improvement of the BER between the FG and the CG can be attributed to the fact that the CG achieved convergence with fewer messages passed than the FG, as given in Table 6.2. This is so since the CG can infer more information with each message since its messages are not limited to a scope of only one symbol. Fewer messages in a loopy graph also implies that less biasing can occur due to the loops in the graph.

The superior BER of the JT is due to the fact that we are able to apply exact inference on the graph, as opposed to approximate inference in the cases of the FG and the CG. As stated previously, approximate inference only allows local optima that can only be found using an iterative procedure. This process is not guaranteed to obtain the optimal solution after decoding. On the other hand, exact inference has a single optimum that can be found in a finite number if steps. Therefore, we will be able to obtain the exact solution for the JT, i.e., the best possible solution given the information available. This is not the case for the FG and the CG.

Unfortunately, the improvements gained by the CG and JT comes at a price. As shown in Table 6.2, both these graphs' computational complexities are significantly greater than that of the FG, assuming decoding is done using the tanh-rule BP when decoding the FG.

Table 6.2: This table shows the order of complexity of each PGM with its associated decoding algorithm. N is the average number of messages passed and $|\text{Scp}(\phi_B)|$ is the average size of the largest cluster in the graph.

PGM	Decoding	Complexity	N	$\lvert\text{Scp}(\phi_B)\rvert$
FG	Tanh-rule BP	$O(N)$	92.4k	24
FG	Loopy BP	$O(N2^{\text{Scp}(\phi_B)})$	92.4k	24
CG	Loopy BP	$O(N2^{\text{Scp}(\phi_B)})$	45.4k	24
JT	Exact BP	$O(2^{\text{Scp}(\phi_B)+1})$	800	39

As expected, the computational complexity of the JT is the greatest of all three graphs. This is due to the Variable Elimination (VE) algorithm that generates larger clusters. Since the computational complexity increases in a manner that is exponentially proportional to the size of the largest cluster, the computational complexity of the JT quickly exceeds that of the FG and the CG.

Table 6.2 also shows the advantages the tanh-rule BP provides. That is, since more messages are passed before convergence in a FG than in a CG, and should loopy BP be applied to both graphs, then the computational complexity of the FG will exceed that of the CG. However, significantly decreased computational complexity of the tanh-rule BP allows the computational complexity of the FG to be superior to that of the CG.

6.7 Summary

This chapter shows the results of the simulations done to test and confirm the topics and theories discussed within this thesis.

Simulations of Raptor code using loopy BP and the FG over the BEC, the BSC, and the BAWGNC were done and their results are given in Section 6.2. The Raptor code used for these simulations is the R10 code with a source data size of 1kb. The error probabilities of the channels were chosen such that their channel capacities were all at $\mathcal{R} = 0.5$. It was found that in all 3 cases the Raptor code has a BER drop-off close to the channel capacity, showing that it is a viable code to use for all 3 channel models.

It was shown in Section 6.3 that the transmission rate of the Raptor code approaches a point close to the channel capacity as the number of input symbols $\mathcal{K}$ goes to infinity. That is, the transmission rate of the Raptor code is close to the channel capacity for large values of $\mathcal{K}$.

In Section 6.4 we showed that the BER of the Raptor code was not limited by an error floor, whereas the LT code was. For these simulations the BSC with a channel capacity of $\mathcal{R} = 0.5$ was used and the tanh-rule loopy BP was applied to the FG to decode both the R10 and LT codes. A source data set of 1kb was used. These simulations illustrate that the Raptor code is an improvement over the LT code.

The graph-altering algorithms, TEP and ID, were tested on the Raptor

code and compared against the tanh-rule loopy BP algorithm, for which the results may be found in Section 6.5. These test simulations were run over the BSC, using FGs to decode the R10 code with a source data set of 1kb. It was found that both the ID and TEP algorithms showed improvements in their BER with respect to the tanh-rule loopy BP algorithm. The TEP is considered preferable due to its lower computational complexity compared to that of ID.

The BERs of the FG, CG, and JT when applied to a Raptor code were compared in Section 6.6. These simulations were done at the channel capacity of $\mathcal{R} = 0.5$. A Raptor code with an LDPC precode and data set of 100 bytes was used. In Fig. 6.5 it can be seen that the BER of the CG is better overall than that of the FG, and that of the JT is the best of the 3.

The computational complexities of the FG, CG, and JT when applied to a Raptor code were given in Table 6.2. This was done by comparing the number of messages passed before successful decoding was achieved. These simulations were done at the channel capacity of $\mathcal{R} = 0.5$. A Raptor code with a LDPC precode and data set of 100 bytes was used. It was found that the computational complexity of the JT exceeds that of the CG, where the CG exceeds that of the FG.

Chapter 7

Conclusion and Recommendations

7.1 Conclusion

Considering that most of the work on the Raptor code is focused on its development for the BEC model, it was our objective to investigate the application of the Raptor code on other channel models. In addition we aimed to apply the latest PGM architectures and decoding techniques to the Raptor code and compare their differences in performance and complexity for this application.

Simulations of the standardised R10 Raptor code as applied to the BEC, BSC, and BAWGNC models using FGs and tanh-rule loopy BP were done. These simulations resulted in similar BER drop-offs, indicating that the Raptor code is a viable method of ensuring reliable communication for all three channel models.

It was found that, as the number of input symbols $\mathcal{K}$ goes to infinity, the BER drop-off of the R10 code approaches a point close to the channel capacity. Since both the random linear fountain code and the R10 code show similar behaviour, it is safe to assume that other Raptor codes would do so as well,, e.g., the RQ code. Accordingly, we may conclude that the transmission rate of the Raptor code is close to the channel capacity for large values of $\mathcal{K}$.

The BER of the Raptor code is not limited by an error floor and the LT code is, as was confirmed through simulations. Since the Raptor code is designed to add little to no computational complexity to the BP decoding algorithm, it can be concluded that the Raptor code represents an improvement over the

LT code.

The graph-altering algorithms, TEP and ID, were tested on the standardised R10 Raptor code and compared against the tanh-rule loopy BP algorithm. It was found that both the ID and TEP algorithms showed improvements in their BER with respect to the tanh-rule loopy BP algorithm. The TEP is considered preferable due to its lower computational complexity compared to that of ID. However, both graph-altering algorithms show only a small BER improvement. Thus the added computational overhead caused by restarting the BP algorithm after each iteration of these algorithms may negate the advantages they present.

Each of the three PGMs assessed in this thesis has its practical advantages and disadvantages. Of the three PGMs, the FG has the worst BER curve; however, it does allow for the application of the tanh-rule BP algorithm that significantly reduces the computational complexity of its decoding.

The tanh-rule BP can unfortunately not be applied to the CG or the JT. Thus, even though the CG has a better BER curve than that of the FG, it comes at the cost of greater computational complexity. Finally, the JT has the best BER curve of the PGMs covered in this thesis. However, since the VE algorithm used to construct the JT results in larger clusters than that of the CG or the FG, it also has a significantly greater computational complexity.

It therefore cannot be decisively stated that one PGM is better than the other. Rather, a balance between the decoding success rate and the computational complexity must be maintained. That is, for example, in an environment where computational resources are scarce, the FG may be preferable. In an environment where the decoding success rate is of the highest importance, the JT may be preferable.

Regardless, it has been demonstrated that the Raptor code is an excellent solution for broadcasting applications over not only the BEC, but also the BSC and BAWGNC. Furthermore, PGMs and the BP algorithm, such as those covered in this thesis, are ideal for decoding fountain code and other Forward Error Correction (FEC) codes.

7.2 Recommendations

The fields of Information theory, error control coding, and PGMs are extensive and many theories included in these fields are not covered in this work. However, many of these theories may prove to be beneficial when applied to Raptor codes. Recommendations for possible improvements on this work are as follows:

- Apply the standardised RQ code to the BSC and BAWGNC and assess its performance as compared to that of the BEC.
- Design precodes and degree distributions for the Raptor code applied over the BSC and BAWGNC. This may be done using density evolution [9].
- Test the BERs of the Raptor code over the BSC and BAWGNC with precodes and degree distributions specifically designed for these channels.
- Express and quantify the information passed in a general message for each PGM covered. This will help verify the BER improvements found on the CG and JT when compared to the FG.
- Apply the Raptor code to other channel models to investigate its viability for such environments - for example, channel models with memory.
- Compare Raptor codes BER with other of other FEC codes such as the LDPC code or convolution codes.
- Adapt the tanh-rule BP to allow for messages that are not limited to a single symbol. This may be extremely advantageous since it may significantly reduce the computational complexity of BP over PGMs, such as the CG.
- Compare the performance of the different message-passing schedules.
- Compare the performance of the different cost functions when constructing a JT by means of the VE algorithm.
- Apply damping to the loopy BP messages. [1]

Appendices

Appendix A

Proofs for channel capacities

A.1 Proof of Theorem 2.2 [2, p. 158]

We assume the general input Probability Distribution (PD) $\mathcal{P}_X = \{p_0, p_1\}$ for the input ensemble X, a output ensemble Y and an erasure probability of ϵ. For the Binary Erasure Channel (BEC), x is only uncertain when erasure $y =?$ is received, therefore from (2.3.2)

$$H(X|Y) = \sum_{y \in \mathcal{A}_Y} P(y)H(X|y) = P(y =?)H(X|y =?)$$

Furthermore, we know that $P(y =?) = p_0\epsilon + p_1\epsilon = \epsilon$ and $P(\hat{x}|y =?) = P(\hat{x})$. Thus,

$$H(X|Y) = \epsilon H_2(p_0)$$

It then follows that

$$\begin{aligned} Cap(\mathcal{C}_{BEC}) &= max_{\mathcal{P}_X}\Big(H(X) - H(X|Y)\Big) \\ &= max_{\mathcal{P}_X}\Big(H_2(p_0) - \epsilon H_2(p_0)\Big) \\ &= max_{\mathcal{P}_X}\Big((1-\epsilon)H_2(p_0)\Big) \end{aligned}$$

The binary entropy function has a maximum when $\mathcal{P}_X = \{0.5, 0.5\}$, which leads us to conclude

$$Cap(\mathcal{C}_{BEC}) = 1 - \epsilon$$

A.2 Proof of Theorem 2.3 [2, p. 158]

We assume the general case input PD $\mathcal{P}_X = \{p_0, p_1\}$ and a bit flip probability of ρ. According to figure 2.4, we will have

$$H(X|Y) = p_0 H_2(\rho) + p_1 H_2(\rho) = H_2(\rho)$$

It thus logically follows that

$$\begin{aligned} Cap(\mathcal{C}_{BSC}) &= max_{\mathcal{P}_X}\Big(H(X) - H(X|Y)\Big) \\ &= max_{\mathcal{P}_X}\Big(H_2(p_0) - H_2(\rho)\Big) \end{aligned}$$

As mentioned before, the binary entropy function has a maximum when $\mathcal{P}_X = \{0.5, 0.5\}$, therefore

$$Cap(\mathcal{C}_{BSC}) = 1 - H_2(\rho)$$

A.3 Proof of Theorem 2.4 [3]

Given the binary input ensemble X and the Gaussian noise $\mathbf{n} = N(0, {\sigma_n}^2)$, the output ensemble is

$$Y = X + \mathbf{n}$$

Since y is a continuous random variable, its entropy is calculated using the differential entropy function defined as

$$H(Y) \stackrel{\text{def}}{=} E\left[log_2 \frac{1}{P(y)}\right] = \int_{\mathcal{A}_y} P(y) log_2 \frac{1}{P(y)} \mathrm{d}y$$

where $\mathcal{A}_y = \{-\infty, \infty\}$. From (2.3.2) it also follows that

$$H(Y|X) = \int_{-\infty}^{\infty} \sum_{x \in \mathcal{A}_X} P(x,y) log_2 \frac{1}{P(x|y)} \mathrm{d}y$$

To simplify the calculations, we write the mutual information in the form of the Kullback-Leibler distance function as follows

$$\begin{aligned} I(Y;X) &= H(Y) - H(Y|X) \\ &= \int_{-\infty}^{\infty} P(y) log_2 \frac{1}{P(y)} \mathrm{d}y - \int_{-\infty}^{\infty} \sum_{x \in \mathcal{A}_X} P(x,y) log_2 \frac{1}{P(x|y)} \mathrm{d}y \\ &= \int_{-\infty}^{\infty} \sum_{x \in \mathcal{A}_X} P(y|x) P(x) log_2 \frac{P(x|y)}{P(y)} \mathrm{d}y \\ &= \int_{-\infty}^{\infty} \sum_{x \in \mathcal{A}_X} P(y,x) log_2 \frac{P(x,y)}{P(y)P(x)} \mathrm{d}y. \end{aligned}$$

Then we have

$$\begin{aligned}
I(Y;X) &= \int_{-\infty}^{\infty} \sum_{x\in\mathcal{A}_X} P(y|x)P(x) log_2 \frac{P(x|y)}{\sum_{x'\in\mathcal{A}_X} P(y|x')P(x')} \mathrm{d}y \\
&= \frac{1}{2}\int_{-\infty}^{\infty} P(y|A) log_2 \frac{P(y|A)}{\frac{1}{2}\Big(P(y|A)+P(y|-A)\Big)} \\
&\quad + P(y|-A) log_2 \frac{P(y|-A)}{\frac{1}{2}\Big(P(y|A)+P(y|-A)\Big)+} \mathrm{d}y \\
&= \frac{1}{2}\int_{-\infty}^{\infty} P(y|A) log_2 P(y|A) + P(y|-A) log_2 P(y|-A) \\
&\quad - \Big(P(y|A)+P(y|-A)\Big) log_2 \Big(\frac{1}{2}\Big(P(y|A)+P(y|-A)\Big)\Big) \mathrm{d}y \\
&= \frac{1}{2}\Bigg(- H(Y) - \int_{-\infty}^{\infty} \Big(P(y|A)+P(y|-A)\Big)\times \\
&\qquad\qquad log_2 \Big(\frac{1}{2}\Big(P(y|A)+P(y|-A)\Big)\Big) \mathrm{d}y \Bigg)
\end{aligned}$$

Since the definition of the variance is $\sigma = E[(y-\mu)^2]$, we may also express $H(Y)$ as

$$\begin{aligned}
H(Y) &= -E\Big[\sum_{x\in\mathcal{A}_X} log_2 P(y|x)\Big] \\
&= -E\left[log_2 \frac{1}{\sqrt{2\pi{\sigma_n}^2}} e^{-\frac{(y-A)^2}{2{\sigma_n}^2}} + log_2 \frac{1}{\sqrt{2\pi{\sigma_n}^2}} e^{-\frac{(y+A)^2}{2{\sigma_n}^2}}\right] \\
&= -E\left[log_2 \frac{1}{2\pi{\sigma_n}^2} e^{-\frac{(y-A)^2+(y+A)^2}{2{\sigma_n}^2}}\right] \\
&= log_2(2\pi{\sigma_n}^2) + \frac{1}{2{\sigma_n}^2}\Big(E\Big[(y-A)^2\Big] + E\Big[(y+A)^2\Big]\Big) log_2(e) \\
&= log_2(2\pi e{\sigma_n}^2).
\end{aligned}$$

Therefore, we conclude that

$$Cap(\mathcal{C}_{BAWGNC}) = -\int_{-\infty}^{\infty} g(y,{\sigma_n}^2) log_2\Big(g(y,{\sigma_n}^2)\Big)\mathrm{d}y - \frac{1}{2} log_2(2\pi e{\sigma_n}^2)$$

where

$$g(y,A,{\sigma_n}^2) = \frac{1}{2}\frac{1}{\sqrt{2\pi{\sigma_n}^2}}\left(\exp\left(-\frac{(y-A)^2}{2{\sigma_n}^2}\right) + \exp\left(-\frac{(y+A)^2}{2{\sigma_n}^2}\right)\right).$$

Appendix B

The universality of the Raptor code

This section gives a compact overview of the work done by Etesami and Shokrollahi in [17], specifically focusing on the universality of the Raptor code for Binary Erasure Channels (BECs), Binary Symmetric Channels (BSCs) and Binary Additive White Gaussian Noise Channels (BAWGNCs).

We start by defining an expression of the expected entropy of the output symbols when using Belief Propagation (BP) decoding as is explained in Section 4.6. In [17] it is proven that this expectation is

$$E(\mathcal{C}) \stackrel{\text{def}}{=} \int_{-\infty}^{\infty} \tanh\left(\frac{x}{2}\right) g(x)\mathrm{d}x$$

where $g(x)$ is the Probability Density Function (PDF) of the Log-Likelihood Ratio (LLR) of the channel. Etesami and Shokrollahi continues to show that

$$\begin{aligned} E(\mathcal{C}_{BEC}) &= 1-\epsilon \\ E(\mathcal{C}_{BSC}) &= (1-2\rho)^2 \\ E(\mathcal{C}_{BAWGNC}) &= \frac{1}{2\sqrt{\pi m}} \int_{-\infty}^{\infty} \tanh\left(\frac{x}{2}\right) e^{\frac{(x-m)^2}{4m}} \mathrm{d}x \end{aligned}$$

with $m = 2/{\sigma_n}^2$. Using this, along with the capacities of the Binary Input Memoryless Symmetric Channels (BIMSCs), the following theorem defines a bound on the degree 1 and 2 probabilities of the degree distribution for a

sequence of capacity-achieving Raptor codes.

Theorem 2.1: (Bounds on Ω_1 and Ω_2) [17]
Assume that $\left(k, \mathcal{C}, \Omega(x)\right)$ is a sequence of capacity-achieving Raptor codes for the BIMSC $\mathcal{C}$, with $\mathcal{M} \geq 1$. Then, given that $E(\mathcal{C}) \neq 0$ and $Cap(\mathcal{C}) \neq 0$, we have:

$$\lim_{\mathcal{M}\to\infty} \Omega_1 = 0 \quad \text{and} \quad \Omega_1 > 0 \ \ \forall \ \mathcal{M}$$

$$\lim_{\mathcal{M}\to\infty} \Omega_2 = \frac{1}{2}\frac{Cap(\mathcal{C})}{E(\mathcal{C})}$$

Proof: The proof for this may be found in [17].

By defining $\Omega_2(\mathcal{C})$ such that

$$\Omega_2(\mathcal{C}) \stackrel{\text{def}}{=} \frac{1}{2}\frac{Cap(\mathcal{C})}{E(\mathcal{C})}$$

we have the following:

$$\Omega_{\text{BEC}}(2) = \frac{1}{2}$$

$$\Omega_{\text{BSC}}(2) = \frac{1}{2}\frac{1 - H_2(\rho)}{(1-2\rho)^2}$$

$$\Omega_{\text{BAWGNC}}(2) = \frac{1}{2}\frac{-\int_{-\infty}^{\infty} g(y, A, {\sigma_n}^2) log_2\Big(g(y, A, {\sigma_n}^2)\Big)\mathrm{d}y - \frac{1}{2} log_2(2\pi e {\sigma_n}^2)}{\frac{1}{2\sqrt{\pi m}}\int_{-\infty}^{\infty} \tanh\left(\frac{x}{2}\right) e^{\frac{(x-m)^2}{4m}}\mathrm{d}x}$$

where A is the signal amplitude, ${\sigma_n}^2$ is the variance of the channel noise, $m = 2/{\sigma_n}^2$, and

$$g(y, A, {\sigma_n}^2) = \frac{1}{2}\frac{1}{\sqrt{2\pi{\sigma_n}^2}}\left(\exp\left(-\frac{(y-A)^2}{2{\sigma_n}^2}\right) + \exp\left(-\frac{(y+A)^2}{2{\sigma_n}^2}\right)\right).$$

The proof for these equations is extensive and includes substantial theory not included in, nor relevant to the main focus of this thesis. Therefore, the proof is not include here and the reader is referred to [17].

These equations are depicted in Fig. 2.19. We see that $\Omega_{\text{BEC}}(2)$ is a constant for any value of ϵ, supporting the fact that Raptor codes are universal for BECs. Unfortunately, we see that this is not the case for BSCs and BAWGNCs, thus concluding that Raptor codes are not universal for these channels.

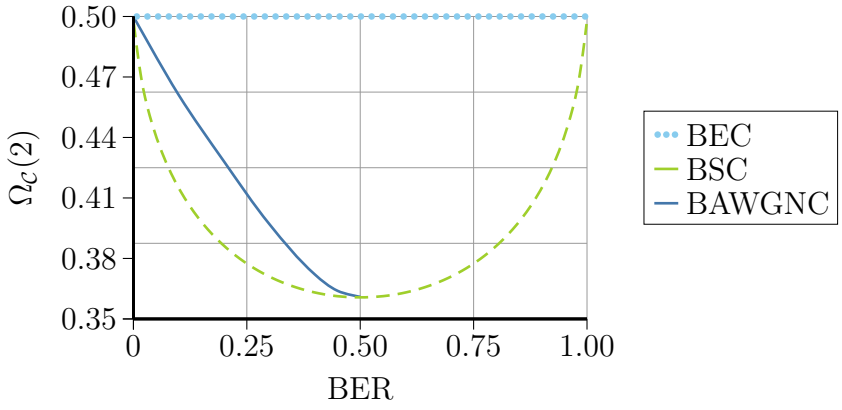


Figure B.1: The bounds on $\Omega_{\mathcal{C}}(2)$ for the BEC, BSC, and BAWGNC. We see that this is a constant for the BEC, an indication that Raptor codes are universal for erasure channels. However, for the BSC and BAWGNC, this is not the case and thus we conclude that the Raptor code is not universal for these channels.

However, the following properties may be derived from Fig. 2.19:

- $0 \leq \Omega_{\mathcal{C}}(2) \leq \frac{1}{2}$
- $\Omega_{\text{BEC}}(2) = \frac{1}{2}$
- $\Omega_{\text{BAWGNC}}(2) \geq \Omega_{\text{BSC}}(2) \geq \frac{1}{\ln(16)}$
- $\lim_{\rho \to \frac{1}{2}} \Omega_{\text{BSC}}(2) = \lim_{\sigma \to \infty} \Omega_{\text{BAWGNC}}(2) = \frac{1}{\ln(16)}$.

Appendix C

Helpful mathematical tricks

C.1 Managing with extremely small or large values

Each step of the process of message passing causes the messages to become a product of more and more functions. This in turn causes the messages to become ether very large or very small (especially when working with normalised distributions). In fact, these message quickly extend what a average computer's numerical system can store.

To overcome this issue it is necessary to work in the logarithmic domain [2]. This is already the case for Factor Graphs (FGs) when using the tanh-rule; for the Cluster graph (CG) and Junction Tree (JT) we need to adapt (4.7.1) such that

$$\begin{aligned}\lambda_{\phi_m\to\phi_n}(\mathbf{x}_m \cap \mathbf{x}_n) &= L\left(\prod_{\substack{\phi_i = \\ \boldsymbol{\phi}_m\setminus\phi_n}} \mu_{\phi_i\to\phi_m}(\mathbf{x}_i \cap \mathbf{x}_m)\right) \\ &= \sum_{\substack{\phi_i = \\ \boldsymbol{\phi}_m\setminus\phi_n}} \lambda_{\phi_i\to\phi_m}(\mathbf{x}_i \cap \mathbf{x}_m).\end{aligned}$$

The summation in equation (4.7.1) is more difficult as it cannot be done in the logarithmic domain. So we need a way to convert our message back to their original numerical values without extend our computer's number system's range.

We may solve this problem by observing that a scaled set of values retains the same information as the original set, enabling us to scale our values (by a

common factor) to a size inside our system's range. In the logarithmic domain we may do this by utilising a second logarithmic identity, that is

$$log\left(\frac{A}{B}\right) = log(A) - log(B)$$

To find a convenient scaling factor we make a second observation that the value of the log of a sequence is typically just a little larger than the log of the maximum value of the sequence.

To illustrate this, let us consider an arbitrary example where we would like to sum over the sequence $\{s_1, s_2, \ldots, s_N\}$ whose values are outside our system's range. We need

$$S = log\left(s_1 + s_2 + \ldots + s_N\right)$$

Let us assume the sequence is sorted from largest to smallest, then our common factor will be s_1:

$$S = log\left(s_1 + s_2 + \ldots + s_N\right) - log(s_1) + log(s_1)$$
$$S = log\left(1 + \frac{s_2}{s_1} + \ldots + \frac{s_N}{s_1}\right) + log\left(s_1\right)$$

This methodology is often used in practice and often carried out in the *negative* log likelihood domain.

List of References

[1] Koller, D. and Friedman, N.: Probabilistic Graphical Models. 2009.

[2] MacKay, D.J.C.: *Information theory, inference and learning algorithms*. 4th edn. Cambridge University Press, March 2003. Available at: http://www.inference.phy.cam.ac.uk/mackay/itila/.
Available at: http://www.cambridge.org/0521642981

[3] Richardson, T.J. and Urbanke, R.: *Modern Coding Theory*. Cambridge University Press, New York, NY, USA, 2008. ISBN 0521852293, 9780521852296.

[4] Shannon, C.E.: A mathematical theory of communication. *Bell system technical journal*, vol. 27, 1948.

[5] Berrou, C., Glavieux, A. and Thitimajshima, P.: Near shannon limit error-correcting coding and decoding: Turbo-codes. In: *Communications, 1993. ICC 93. Geneva. Technical Program, Conference Record, IEEE International Conference on*, vol. 2, pp. 1064 –1070 vol.2. may 1993.

[6] Richardson, T.J. and Urbanke, R.L.: The capacity of low-density parity-check codes under message-passing decoding. *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 599 –618, feb 2001. ISSN 0018-9448.

[7] Richardson, T.J., Shokrollahi, A. and Urbanke, R.L.: Design of capacity-approaching irregular low-density parity-check codes. *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 619 –637, feb 2001. ISSN 0018-9448.

[8] Shokrollahi, A. and Luby, M.: Raptor codes. *Foundations and Trends in Communications and Information Theory*, vol. 6, no. 3-4, pp. 213–322, 2011.
Available at: http://dx.doi.org/10.1561/0100000060

[9] Luby, M.G., Mitzenmacher, M. and Shokrollahi, A.: Analysis of random processes via and-or tree evaluation. In: *In Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 364–373. 1998.

[10] Luby, M.G.: Lt codes. *Foundations of Computer Science, IEEE Annual Symposium on*, p. 271, 2002. ISSN 0272-5428.

[11] Pakzad, P. and Shokrollahi, A.: Design Principles for Raptor Codes. In: *Proceedings of the IEEE Information Theory Workshop*, pp. 165–169. 2006.

[12] Shokrollahi, A., Lassen, S. and Luby, M.: Multi-stage code generator and decoder for communication system. April 10 2013. EP Patent App. EP20,100,013,232.
Available at: http://www.google.com/patents/EP2315357A3?cl=en

[13] 3GPP TS 26.346: Technical specification group services and system aspects; multimedia broadcast/multicast service; protocols and codecs. June 2005.

[14] Luby, M., Shokrollahi, A., Watson, M., and Stockhammer, T.: Raptor forward error correction scheme for object delivery. *Internet Engineering Task Force, RFC 5053*, September 2007.
Available at: http://tools.ietf.org/html/rfc5053

[15] Singels, R., du Preez, J. and Wolhuter, R.: Soft decoding of raptor codes over awgn channels using probabilistic graphical models. In: *Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2012*. Telkom SA ltd., September 2012.

[16] Moon, T.K.: *Error Correction Coding: Mathematical Methods and Algorithms*. Wiley-Interscience, 2005. ISBN 0471648000.

[17] Etesami, O. and Shokrollahi, A.: Raptor Codes on Binary Memoryless Symmetric Channels. *IEEE Trans. Inf. Theory*, vol. 52, no. 5, pp. 2033–2051, 2006.

[18] MacKay, D.J.C.: Fountain codes. In: *The IEE Seminar on Sparse-Graph Codes*, pp. 1–8. IEE, London, 2004.

[19] Olmos, P., Murillo-Fuentes, J. and andrez Cruz, F.P.: Tree-structure expectation propagation for decoding ldpc codes over binary erasure channels. In: *Information Theory Proceedings (ISIT), 2010 IEEE International Symposium on*, pp. 799 –803. June 2010.

[20] Shokrollahi, A., Lassen, S. and Karp, R.: Systems and processes for decoding chain reaction codes through inactivation. *U.S. Patent number 6 856 263*, February 2005.

[21] Gallager, R.G.: *Information Theory and Reliable Communication*. John Wiley & Sons, Inc., New York, NY, USA, 1968. ISBN 0471290483.

[22] Nasif, A.O. and Karystinos, G.N.: Binary Transmissions over Additive Gaussian Noise: A Closed-Form Expression for the Channel Capacity. In: *Conference on Information Sciences and Systems, The Johns Hopkins University*. March 2005.

[23] Letter frequency. wikipedia, September 2012.
Available at: http://en.wikipedia.org/wiki/Letter_frequency

[24] Guede, M.A.: *Optimization of the Belief Propagation algorithm for Luby Transform decoding over the Binary Erasure Channel*. Master's thesis, Delft University of Technology, Augustus 2011.

[25] Yeung, R.: A new outlook on Shannon's information measures. *IEEE Trans. Inf. Theory*, vol. 37, no. 3, pp. 466 –474, may 1991. ISSN 0018-9448.

[26] Neubauer, A., Freudenberger, J. and Kuhn, V.: *Coding Theory: Algorithms, Architectures and Applications*. John Wiley & Sons, 2008. ISBN 9780470519820.
Available at: http://books.google.co.za/books?id=yz_OsVx_I28C

[27] Stern, H.P.E., Mahmoud, S.A. and Stern, L.E.: *Communication systems: analysis and design*. v. 1. Pearson Prentice Hall, 2004. ISBN 0130402680, 9780130402684.
Available at: http://books.google.co.za/books?id=mXVGAAAAYAAJ

[28] Clark, G.C.J. and Cain, J.B.: *Error-correction coding for digital communications*, vol. Second Printing. Plenum Press, New York, August 982.

[29] Gallager, R.G.: Low-Density Parity-Check Codes. 1963.

[30] Johnson, S.J.: Introducing Low-Density Parity-Check codes. Tech. Rep., Department of Electrical and Computer Engineering, University of Newcastle, Australia, Unknown.

[31] Wiid, R.: *Implementation of a Protocol and Channel Coding Strategy for use in Ground-Satellite Applications*. Master's thesis, University of Stellenbosch, March 2012.

[32] Luby, M.G., Mitzenmacher, M., Shokrollahi, A. and Spielman, D.A.: Improved Low-Density Parity-Check codes using irregular graphs. *IEEE Trans. Inf. Theory*, vol. 47, pp. 585–598, 2001.

[33] Byers, J.W., Luby, M., Mitzenmacher, M. and Rege, A.: A digital fountain approach to reliable distribution of bulk data. *SIGCOMM Comput. Commun. Rev.*, vol. 28, no. 4, pp. 56–67, October 1998. ISSN 0146-4833.
Available at: http://doi.acm.org/10.1145/285243.285258

[34] Byers, J.W., Luby, M. and Mitzenmacher, M.: A digital fountain approach to asynchronous reliable multicast. *IEEE J. Sel. Areas Commun.*, vol. 20, pp. 1528–1540, 2002.

[35] Luby, M.G.: Information additive code generator and decoder for communication systems. US Patent no. 7812743, October 2010.

[36] Russell, J.S.: Report on waves. In: *Report of the fourteenth meeting of the British Association for the Advancement of Science*, pp. 311–390. London, 1845. Plates XLVII-LVII.

[37] Gross, J.L. and Yellen, J.: *Handbook of graph theory.* CRC press LLC, 2004.

[38] Frey, B.J. and Kschischang, F.R.: Probability propagation and iterative decoding. In: *Proceedings of the 34th Allerton Conference on Communications, Control and Computing*, pp. 482–493. 1996.

[39] Barber, D.: *Bayesian Reasoning and Machine Learning.* Cambridge University Press, 2012.

[40] Frey, B.: Extending factor graphs so as to unify directed and undirected graphical models. In: *Proceedings of the Nineteenth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pp. 257–264. Morgan Kaufmann, San Francisco, CA, 2003.

[41] Kschischang, F.R., Frey, B.J. and Loeliger, H.-A.: Factor graphs and the sum product algorithm. *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 498–519, February 2001.

[42] Zhu, X.: Advanced natural language processing. 2010.

[43] Hagenauer, J., Offer, E. and Papke, L.: Iterative decoding of binary block and convolutional codes. *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, 1996.

[44] Qualcomm: Raptorq. Tech. Rep., Qualcomm Incorporated, 2010.
Available at: www.qualcomm.com/raptor

[45] Doran, R.W., Calude, C.S., Stefanescu, G. and Zimand, M.: The gray code. *Journal of Universal Computer Science*, vol. 13, no. 11, Nov 2007.

[46] Palanki, R., Palanki, R., Yedidia, J.S. and Yedidia, J.S.: Rateless Codes on Noisy Channels. In: *in Proc. Int. Symp. Inform. Theory*, p. 37. 2004.